NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

Controlling Functional Complexity for Overfitting Avoidance in Genetic Programming

Inês Marcão Cortes Magessi

Master Thesis

presented as partial requirement for obtaining a Master's Degree in Data Science and Advanced Analytics

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

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

Controlling Functional Complexity for Overfitting Avoidance in Genetic Programming

by

Inês Marcão Cortes Magessi

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

Supervised by

Leonardo Vanneschi, PhD, NOVA Information Management School

July, 2024

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 acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

[Lisbon, 15 of July 2024]

Controlling Functional Complexity for Overfitting avoidance in Genetic Programming

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.18) (J. M. Lourenço, 2021).

Acknowledgements

Firstly, I want to thank my supervisor, Professor Leonardo Vanneschi, for everything he has taught me. I have learned a lot of valuable lessons from him, beyond just academics. His care for his students and his dedication to their success are truly inspiring, and I am grateful for his guidance and friendship, which were essential in navigating both the challenges and successes of this work. I will always look up to him as a model of both mentorship and personal excellence.

I am equally grateful to Professor Sara Silva for her stimulating intellectual discussions and for her guidance during this year. Her insights and feedback were invaluable in shaping the direction of this thesis.

Additionally, I would like to thank my family for their constant support for my studies and future aspirations. Their encouragement and love has been a consistent source of strength and ambition. I also want to extend my gratitude to my friends for always being curious about my work, for continuously challenging me to stay focused and for giving me the necessary breaks to recharge.

Lastly, I want to give a special thanks to João for his unconditional support and for our stimulating and endless discussions, which have continuously fueled my motivation and curiosity for the topic of this work.

”

“Nature is pleased with simplicity. And nature is no dummy.”

— Isaac Newton

Resumo

A Programação Genética (PG) é uma técnica versátil no campo da Computação Evolucionária que oferece soluções para uma ampla variedade de problemas. Este trabalho foca-se numa aplicação comum da PG - a Regressão Simbólica (RS) - que tem como objetivo descobrir funções matemáticas que descrevam as relações entre variáveis de *input* e *output* de um *dataset*. Embora a PG se destaque pela sua capacidade de evoluir expressões diversas sem tamanho ou forma pré-definida, um desafio central é o *sobreajuste* (ou *overfitting*, como é normalmente chamado), fenómeno que se verifica quando as funções se ajustam demasiado aos dados de treino, comprometendo a sua capacidade de generalização em novos dados.

Apesar das técnicas de regularização tradicionalmente usadas para combate de *overfitting* estarem amplamente estudadas na literatura, estas são difíceis de aplicar diretamente à PG devido à sua estrutura flexível e ausência de otimização de parâmetros. Posto isto, este estudo propõe uma abordagem inovadora para minimizar o *overfitting* em PG, que se baseia numa otimização dupla ao longo da evolução: a minimização do erro e a penalização da complexidade funcional das expressões evoluídas, recorrendo a mecanismos de seleção multi-objetivo. Enquanto a minimização do erro é um objetivo comum em PG, a penalização da complexidade funcional é um passo extra que procura evitar a evolução de expressões excessivamente complexas e sobreajustadas. A medida de complexidade funcional utilizada neste estudo aproxima a curvatura de uma função, refletindo a sua tendência para sobreajustar os dados de treino.

Resultados experimentais em oito conjuntos de dados de RS demonstram a eficácia das duas variantes do método proposto na redução de *overfitting*, por comparação com os resultados de referência da PG padrão. O estudo realça a importância de equilibrar a redução de complexidade com a capacidade preditiva nos modelos evoluídos pela PG, de forma a garantir que tanto funções precisas como simples são selecionadas. Além disso, são analisados os impactos de diferentes hiperparâmetros numa das variantes do método proposto, assim como são analisadas várias características das funções evoluídas, como a sua curvatura e tamanho. É também estabelecida, de modo formal, uma correlação entre a complexidade funcional e o *overfitting*, e os benefícios do método

proposto para a interpretabilidade dos modelos e seleção de variáveis e redução de *bloat* são discutidos.

Palavras-chave: Programação Genética, Regressão Simbólica, Overfitting, Complexidade Funcional, Optimização Multi-Objectivo

Abstract

Genetic programming (GP) is a versatile technique within the field of Evolutionary Computation (EC), offering solutions to a wide range of problems. This work focuses on Symbolic Regression (SR), a common application of GP that aims to discover mathematical functions describing the relationships between *input* and *target* variables of a given *dataset*. While GP stands out for evolving diverse functions with no constraints in size or shape, a key challenge is *overfitting*, where models become too adjusted to the training data, compromising their generalization to new, unseen data.

Although traditional regularization techniques are widely studied in the literature, they are challenging to apply directly to GP due to its flexible structure and lack of parameter optimization. Therefore, this work proposes a novel approach to mitigate *overfitting* in GP, involving a dual optimization process throughout evolution: minimizing error and penalizing the functional complexity of expressions using multi-objective selection mechanisms. While error minimization is a common goal in GP, penalizing functional complexity is an additional step aimed at avoiding overly complex functions. The functional complexity measure used in this study approximates the curvature of an expression, reflecting its tendency to overfit the training data.

Experimental results on eight SR datasets demonstrate the effectiveness of two variants of the proposed method in reducing *overfitting*, as evidenced by a comparison to the baseline results of Standard GP (StdGP). The study explores and emphasizes the importance of balancing complexity reduction with overall predictive accuracy of evolved models, ensuring the selection of both accurate and simple functions. Additionally, the impact of different hyperparameters on the proposed method is analyzed, along with various characteristics of the evolved functions, such as their curvature and size. Finally, a formal correlation between functional complexity and overfitting is established, and the benefits of the proposed method for model interpretability, feature selection and *bloat* reduction are discussed.

Keywords: Genetic Programming, Symbolic Regression, Overfitting, Functional Complexity, Multi-Objective Optimization

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI	Artificial Intelligence
DT	Double Tournament
EA	Evolutionary Algorithm
EC	Evolutionary Computation
GA	Genetic Algorithm
GP	Genetic Programming
ML	Machine Learning
MMOTS	Modified Multi-Objective Tournament Scheme
SBC	Slope-Based Complexity
SR	Symbolic Regression
StdGP	Standard GP

1

Introduction

Genetic Programming (GP) is a powerful technique within the field of Evolutionary Computation (EC). Koza defined GP as a method to find a computer program of unspecified size and shape to solve, or approximately solve, a problem (Koza, 1992). Its versatility is evident in its successful applications across various domains, including regression (Koza, 1992), control system design (Bourmistrova & Khantsis, 2007), data mining (Cavaretta & Chellapilla, 1999), and optimization (Grimes, 1995).

GP is inspired by the principles of Darwinian evolution (Darwin, 1964), where a population of candidate solutions evolves over generation to find the optimal solution to a given problem. Traditionally, each candidate solution in GP is represented using a tree structure: nodes denote primitive functions such as mathematical operators, while leaves represent operands such as variables or constants, although other representations are also possible. The population evolves through genetic operators like mutation and crossover, which modify the trees to create new candidate solutions. The fitness of each candidate solution is evaluated based on its performance on a given problem, and the fittest ones are selected to reproduce and form the next generation (Koza, 1992).

One of the most common applications of GP is Symbolic Regression (SR), a type of regression analysis aimed at discovering a mathematical expression that best describes the relationships between independent variables and a *target* within a given dataset (Tohme et al., 2023). In SR problems solved using GP, candidate solutions are typically represented as trees that encode mathematical expressions. These expressions evolve to optimize the mapping between *input* and *output* variables in the dataset. GP offers several advantages for SR, including its ability to automatically generate a variety of nonlinear functions, perform variable selection implicitly without added cost, and disregard assumptions about the independence of input variables, thereby reducing the problem's dimensionality (Vladislavleva et al., 2009).

Another GP's main strength lies in its ability to evolve virtually any function, given some set of pre-defined variables and primitive functions - the terminal and non-terminal sets. This characteristic sets it apart from other machine learning paradigms,

as it does not impose constraints on the size or shape of the functions it can create. While traditional models like linear regression struggle with complex relationships, GP can explore these patterns, making it a valuable tool in domains with complex or highly non-linear underlying structures.

However, one of the key challenges in GP, as in many machine learning techniques, is *overfitting*. This phenomenon arises when a model becomes too adjusted to the training data, capturing noise and irrelevant patterns rather than the data's underlying structure (Bishop, 2006). GP's capability to evolve diverse functions without constraints on their shape and size can result in highly complex and overfitted models. While GP succeeds at evolving complex functions based on their fit to observed data, this strength can also lead to fitting the training data too precisely, compromising the models' ability to generalize to new, unseen data. Essentially, an overfitted model might memorize the training set rather than learn meaningful patterns, interfering with its effectiveness in generalization. Consequently, selecting GP-evolved models solely based on their fit to training data tends to favor unnecessarily complex models that perform poorly in terms of generalization.

Regularization techniques are commonly employed in machine learning to address *overfitting*. The traditional methods introduce a penalty term to the loss function, aiming to control for the models' complexity and prevent them from fitting the noise in the data. However, applying traditional regularization methods like Lasso (Tibshirani, 2018), Ridge (Hoerl & Kennard, 2000), or Elastic Net (Zou & Hastie, 2005) directly to GP can be challenging due to its lack of a fixed model structure or coefficients to optimize (Vanneschi & Castelli, 2021). Instead, regularization in GP often relies on alternative approaches to penalize overfitted models by ensuring that the evolution considers more than just the fit to the observed data.

This work focuses on the issue of *overfitting* in GP and proposes a novel approach to mitigate it. Central to this approach is the idea of using functional complexity as an approximation for a function's tendency to overfit the training data. The functional complexity measure used in this work quantifies the curvature of a function, which in turn measures its smoothness or roughness (Vanneschi et al., 2010). It is hypothesized that simpler functions are more likely to generalize well to unseen data, while complex functions are more likely to overfit.

Two selection mechanisms enabling Multi-Objective Optimization (Sharma & Chahar, 2022), namely the Double Tournament (DT) (Luke & Panait, 2002a) and a Modified Multi-Objective Tournament Scheme (MMOTS) (Azad & Ryan, 2011), are applied in a novel manner to penalize both the error and the functional complexity of evolved functions. While these individual components - multi-objective optimization and functional complexity - are not new to GP, their combined use in this context represents a novel approach. The goal is to penalize complex functions that may be prone to overfitting the training data, while still ensuring the discovery of accurate models capable of capturing the underlying patterns. These methods are evaluated on eight different

symbolic regression datasets, and are compared with the baseline results of Standard GP (StdGP), where no complexity penalty is imposed. A study on the impact of various hyperparameters on the performance of the proposed methods is also conducted, as well as an in-depth analysis of the evolved functions, such as their curvature and size. Finally, the correlation between functional complexity and generalization ability is formally established, and the benefits of the proposed methods for model interpretability, feature selection and *bloat* reduction are discussed.

This document is organized as follows: A theoretical background on optimization, GP, overfitting, and regularization is provided in Chapter 2. An overview of the related work is detailed in Chapter 3. The proposed approaches to mitigate overfitting in GP are described in Chapter 4. Chapter 5 covers the experimental setup, results, and comparison of the proposed methods. It includes descriptions of the datasets used in Section 5.1, details of the experimental setup in Section 5.2, and the presentation and discussion of the results, impacts of various hyperparameters, and an in-depth analysis of the evolved functions in Section 5.3. Finally, the thesis is concluded in Chapter 6, and future work is discussed.

2

Theoretical Background

2.1 Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that focuses on the development of algorithms and models that allow computers to learn from data and make predictions or decisions without being explicitly programmed to do so (K. P. Murphy, 2012). At its core, machine learning is about empowering machines to learn from data, marking a significant shift in problem-solving approaches across various fields.

As defined by Tom Mitchell, a pioneer in the field, ML is the process by which machines acquire knowledge from experience (Mitchell, 1997). Instead of encoding explicit instructions for every possible scenario, ML introduces a more adaptive approach: providing machines with data and allowing them to identify patterns, extract insights, and ultimately, make informed decisions autonomously.

This approach differs from traditional programming, where developers write deterministic algorithms that define the exact steps a machine should take to solve a problem. These algorithms receive inputs and return the corresponding outputs. In contrast, ML algorithms are given inputs (a dataset) and a task to perform (the outputs), and their job is to find the algorithm that maps inputs to outputs (see Fig. 2.1). This process involves the use of a learning algorithm, where the machine iteratively adjusts its parameters to improve its outputs. The model resulting from this learning process is then used to make predictions on new data (Vanneschi & Silva, 2023).

At the heart of ML is the concept of training data - a set of examples used to train algorithms and build models. This data is composed by a set of instances, also known as *fitness cases*, each characterized by a set of features, the units of information that describe the instances (Vanneschi & Silva, 2023). When training a model using a supervised ML algorithm, the training data is typically divided into training, validation, and test sets. The training set is used to train the model, the validation set is used to tune the model's hyperparameters to make them generalize better to unseen data, and the test set is used to evaluate the model's performance on unseen data (Bishop, 2006).

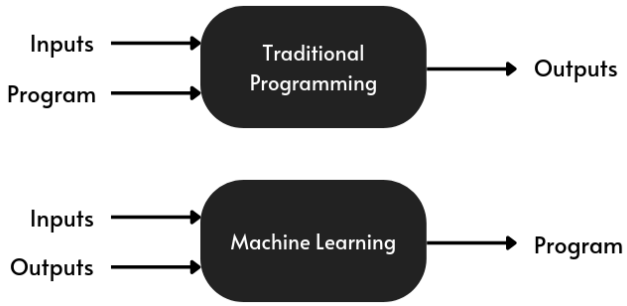


Figure 2.1: Traditional Programming VS Machine Learning

Typical problems in ML are either problems that are too large to be solved by humans or problems that are too complex to be encoded in a traditional algorithm. This field finds application across diverse domains, including healthcare, finance, transportation, and entertainment (Domingos, 2018). From medical diagnostics to personalized recommendations, its impact spans industries, driving innovation and progress.

2.1.1 Learning Methods

ML algorithms can be broadly categorized into three types, each with its own learning method: supervised learning, unsupervised learning, and reinforcement learning.

Supervised Learning Supervised learning is a category of ML where the algorithm learns to map inputs to outputs based on labeled training data (Mohri et al., 2012). The two most common types of supervised learning problems are regression and classification. Regression involves predicting a continuous output variable, such as house prices or the temperature of a city. Classification focuses on predicting a discrete output variable, such as an object's class or a text's sentiment (Hastie et al., 2009).

In this type of learning, a dataset can be formally defined as a set of pairs $(\mathbf{x}, y)$, where $\mathbf{x}$ is an input observation and y is the corresponding output value, also known as *target*. A dataset of size N can be represented as:

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\} \tag{2.1}$$

Generally, each instance can be composed by m features and there are N instances. Therefore, the dataset can be represented as a matrix of size $N \times m$ and a vector Y of size N:

$$D = \left[\begin{array}{cccc|c} x_{11} & x_{12} & \dots & x_{1m} & y_1 \\ x_{21} & x_{22} & \dots & x_{2m} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{Nm} & y_N \end{array}\right]$$

The goal of supervised learning is to obtain a function that maps inputs to outputs, given a set of training examples, such that:

$$\forall i = 1, 2, \dots, N, \quad f(\mathbf{x}_i) = y_i \tag{2.2}$$

A supervised ML algorithm is trained using a labeled dataset, where the correct output is provided for each input. This learning process relies on a pre-defined loss function, used to measure the difference between the predicted output and the actual output (Hastie et al., 2009). The algorithm iteratively adjusts its parameters to minimize the loss function, making the predictions more accurate. Ideally, the loss function should be convex, so that the algorithm converges to the global minimum (Goodfellow et al., 2016). The ultimate goal is that the algorithm learns to generalize from the training examples and make predictions on new, unseen data, where the output is unknown.

Common supervised learning algorithms include linear regression, logistic regression, support vector machines, decision trees, random forests, and neural networks (K. P. Murphy, 2012). These algorithms are used to solve a wide range of problems, from predicting stock prices to classifying images and detecting fraud.

Unsupervised Learning Unsupervised learning is a type of ML where the algorithm learns to identify patterns and relationships in data without being provided with labeled examples (Hastie et al., 2009). The goal of this type of learning is to find hidden structures in the data.

In unsupervised learning, the dataset is composed of input data only, without any corresponding output labels. The algorithm is tasked with discovering the underlying structure of the data, such as grouping similar instances together or identifying patterns that are not immediately apparent (Mohri et al., 2012).

This type of learning is commonly used for tasks such as clustering, dimensionality reduction, and anomaly detection (Alpaydin, 2014). By exploring the data and identifying patterns, unsupervised learning algorithms can provide valuable insights and help uncover hidden relationships in the data. The most well-known unsupervised learning algorithms are K-means Clustering, Hierarchical Clustering, Principal Component Analysis (PCA), and Autoencoders (Celebi & Aydin, 2016). These algorithms are used in problems such as customer segmentation, image compression and anomaly detection.

Reinforcement Learning Reinforcement learning is another type of ML where an agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties (Sutton & Barto, 2018). The goal of reinforcement learning is to learn a policy that maximizes the cumulative reward over time. By exploring the environment and learning from the feedback received, the agent can improve its decision-making over time and learn to perform complex tasks.

Applications of reinforcement learning include game playing, robotics, and autonomous systems (Sutton & Barto, 2018). By learning from experience and adapting to changing environments, reinforcement learning algorithms can achieve impressive results in a wide range of domains.

2.1.2 Evaluation Metrics

Evaluation metrics are used to assess the performance of Machine Learning models. These metrics provide a quantitative measure of how well a model is performing on a given task, such as classification, regression, or clustering (Han et al., 2011). These metrics are usually related to the loss function used to train the model, but they may provide a more interpretable measure of the model's performance.

The choice of evaluation metric depends on the specific problem being addressed and the type of learning algorithm being used. For instance, in classification problems, common evaluation metrics include accuracy, precision, recall, F1 score, and area under the ROC curve (AUC-ROC) (Zaki & Meira, 2020). These metrics provide insights into how well a model is classifying instances into different categories and can help identify areas for improvement. Accuracy measures the proportion of correctly classified instances, precision measures the proportion of true positive predictions among all positive predictions, recall measures the proportion of true positive predictions among all actual positive instances, and the F1 score is the harmonic mean of precision and recall. The AUC-ROC is a measure of the model's ability to distinguish between classes and is commonly used for binary classification problems.

In regression problems, evaluation metrics such as the root mean squared error (RMSE) and mean absolute error (MAE) are commonly used (K. P. Murphy, 2012). These metrics are composed by the sum of the residuals, which are the differences between the predicted values and the actual values, and provide a measure of how well the model is predicting continuous output variables.

The RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{2.3}$$

where y_i is the actual value of the output variable and $\hat{y}_i$ is the predicted value. This metric penalizes large errors more than small errors, making it sensitive to outliers.

Unsupervised learning evaluation metrics are different from supervised learning metrics, as they assess the quality of the model's output without the use of labeled data. These metrics vary based on the specific task. For instance, metrics like the silhouette score, Davies-Bouldin index, and adjusted Rand index are commonly used to evaluate clustering algorithms (Celebi & Aydin, 2016), while metrics like explained variance and reconstruction error are used to evaluate dimensionality reduction algorithms (Tipping & Bishop, 2002).

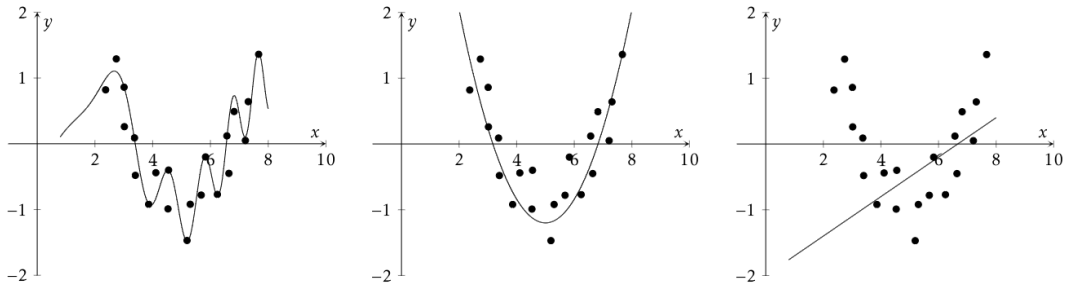
2.1.3 Overfitting and Regularization

When training ML models, it is common to encounter overfitting, a phenomenon where the model performs well on the training data but poorly on unseen data. It occurs when the model is too complex and captures noise in the training data, rather than the underlying patterns (Bishop, 2006). This scenario is undesirable, as the ultimate goal of ML is to build models that learn from a subset of data (the training data) and can be used to make predictions on new data.

When talking about overfitting, it is important to understand the *Bias-Variance Tradeoff* (von Luxburg & Schoelkopf, 2008). *Bias* refers to the error introduced by approximating a real-world problem with a simplified model, while *variance* refers to the error introduced by the model's sensitivity to fluctuations in the training data. High bias can lead to underfitting, where the model is too simple to capture the underlying patterns in the data, while high variance can lead to overfitting, where the model is too complex and captures noise in the training data.

Take the example in Fig. 2.2(a), where a polynomial function is fitted to a set of data points. The function is too complex and captures the noise in the data, resulting in a model that fits the training data very closely. This function will likely not generalize well to unseen data, as it is too sensitive to fluctuations in the training data. On the other hand, in Fig. 2.2(b), the function is a better fit to the data, capturing the underlying patterns without overfitting. Although the error is higher in this example, the model will likely generalize better to unseen data. In Fig. 2.2(c), the function is too simple and fails to capture the underlying patterns, resulting in underfitting. This model will also perform poorly on unseen data, as it is too simple to capture the complexity of the observations.

Regularization is a technique used to prevent overfitting. In some parametrical models, such as linear regression or neural networks, regularization can be applied by adding a penalty term to the loss function that discourages the model from becoming too complex. The penalty term is typically a function of the model's parameters. When using regularization, the loss function penalizes large values of the parameters, in order to shrink them towards zero. In the case of linear regression, L1 regularization (Lasso) (Tibshirani, 2018) adds the absolute value of the parameters to the loss function, making some of them exactly zero, while L2 regularization (Ridge) (Hoerl & Kennard,



(a) Overfitted function (b) Right Fit (c) Underfitting

Figure 2.2: A set of functions that represent different levels of fitting.

2000) adds the square of the parameters, making them smaller. The regularization strength is controlled by a hyperparameter, the penalty factor, typically denoted by λ. By tuning this hyperparameter, the model's complexity can be adjusted to prevent overfitting. The consequence of shrinking the parameters is that the model becomes smoother and less sensitive to fluctuations in the training data. This helps the model generalize better to unseen data and reduces the risk of overfitting.

Figures 2.2(a), 2.2(b) can serve as an example of a non-regularized model and a regularized model, respectively. As it is possible to observe in the first figure, the model is too complex and fits the noise in the data, which is a sign of overfitting. On the other hand, in the second figure, the model follows roughly the same pattern as the data, but is smoother and less sensitive to fluctuations, which is a sign of a well-fitted model that generalizes better to unseen data.

Non-parametrical algorithms can not be regularized in the same way as parametrical ones. However, in algorithms such as decision trees, for example, overfitting can be prevented by limiting the depth of the tree or the number of leaf nodes. These techniques reduce the complexity of the model and prevent it from capturing noise in the training data, improving the its generalization performance.

2.2 Optimization

Optimization is a fundamental concept in ML and AI. It refers to the process of finding the best solution to a problem from a usually large set of possible solutions. This is done by intelligently searching through the solution space to identify the optimal solution(s) that minimize(s) or maximize(s) an objective function (Vanneschi & Silva, 2023).

Usually, optimization problems are NP-complete (Garey & Johnson, 1990), which means that they are computationally hard to solve in polynomial time. This means that optimal solutions cannot be found in a reasonable amount by means of classical

deterministic algorithms. Therefore, optimization algorithms aim to find good approximations of the global optimum in a reasonable amount of time (Vanneschi & Silva, 2023).

An instance of an optimization problem can be formally defined as follows:

$$(S, f) \tag{2.4}$$

where S is the set of all possible solutions, also called *search space*, and $f : S \to \mathbb{R}$ is the objective function that assigns a real value to each solution in S. The goal of optimization is to find the solution $s \in S$ that minimizes or maximizes the objective function $f(s)$.

The No Free Lunch Theorem (Wolpert & Macready, 1997) states that no optimization algorithm is universally better than any other. This means that the performance of an optimization algorithm depends on the specific problem being solved. Different optimization algorithms are designed to tackle different types of problems, such as continuous optimization, combinatorial optimization, and constrained optimization.

The optimization process involves searching through the solution space, which can be mapped to a fitness landscape. The fitness landscape characterizes the relationship between the solutions in the search space and their corresponding fitness values, and can be interpreted as a topographical map of the search space (Jones, 1995). The goal of optimization is to find the global optimum, which is the solution that minimizes or maximizes the objective function across the entire search space.

A fitness landscape represents solutions surrounded by their neighbor solutions, where the fitness value of each solution is represented by a height. The landscape can be visualized as an N-dimensional surface, where N-1 axes represent the solution space the Nth axis represents the fitness value. The fitness landscape can have multiple peaks and valleys, representing local optima and global optima, respectively, in case the objective is to maximize the fitness value. The goal of optimization is to find the global optimum, which is the highest peak in the landscape (Goldberg, 1988). However, as fitness landscapes are usually not convex, it is often difficult to find the global optimum, and optimization algorithms aim to find a good approximation of it.

Multiple optimization algorithms have been developed to tackle different types of optimization problems. These include hill climbing, simulated annealing, particle swarm optimization, genetic algorithms, genetic programming and many others (Vanneschi & Silva, 2023). Each algorithm has its strengths and weaknesses and is suited to different types of problems.

For instance, parametrical supervised learning algorithms, such as linear regression, logistic regression, and neural networks, are typically trained using optimization algorithms such as gradient descent. These algorithms iteratively adjust the model's parameters to minimize the loss function and improve the model's predictions. In this

case, the fitness landscape is defined by the loss function, and the goal is to find the set of parameters that minimize the loss (Goodfellow et al., 2016).

2.3 Evolutionary Computation

Evolutionary Computation (EC) is a family of algorithms that draws inspiration from the principles of biological evolution to solve optimization problems. They are a family of population-based trial and error problem solvers with a stochastic optimization and meta-heuristic character (Eiben & Smith, 2015). The concept of EC dates to even before computers were invented. In 1948, Alan Turing proposed a method of genetic search, the B-type u-machines. However, the field was only established during the 1950s and 1960s, when three branches of EC were developed: evolutionary strategies, evolutionary programming, and genetic algorithms (Fogel, 1998; Haupt, 2009).

An Evolutionary Algorithm (EA) is an example of an EC algorithm. These algorithms use the principles of natural selection, mutation, and recombination to search through the solution space and find the best solution to a problem. They are inspired in the theory of evolution by Charles Darwin (Darwin, 1964), which states that species evolve over time through the process of natural selection, where individuals with favorable traits are more likely to survive and reproduce.

These algorithms work by maintaining a population of candidate solutions, also known as individuals, which are evaluated based on their fitness. Fitness is a measure of how well each individual perform on the optimization problem. The individuals are then subjected to evolutionary operators, such as crossover, and mutation, to generate new candidate solutions. The process is repeated iteratively, aiming for the population to evolve towards better solutions over time (Eiben & Smith, 2015).

The goal is to explore the search space in an intelligent way, leveraging the principles of natural selection to guide the search towards promising regions of the solution space (Eiben & Smith, 2015). By maintaining a diverse population and allowing for random exploration, evolutionary algorithms can escape local optima and find good solutions to complex optimization problems (Vanneschi & Silva, 2023). They aim at providing high-quality solutions in a reasonable amount of time, even for problems with high dimensionality or non-convex fitness landscapes, despite the fact that they do not guarantee to find the global optimum.

EAs follow the same general structure, which consists of the following steps:

1. Population Initialization: A population of candidate solutions is randomly generated to start the optimization process.

2. Fitness Evaluation: Each individual in the population is evaluated based on its fitness, which is a measure of how well it performs on the optimization problem.

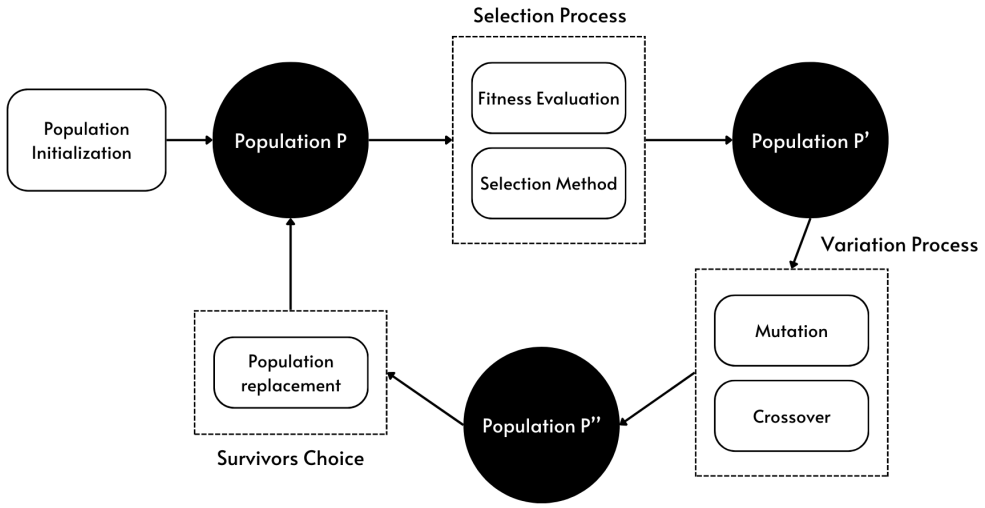


Figure 2.3: Evolutionary Algorithm Process. Inspired by (Vanneschi & Silva, 2023)

3. Selection: Individuals are selected from the population based on their fitness, with fitter individuals being more likely to be selected.

4. Genetic Operators: Selected individuals are subjected to genetic operators, such as crossover and mutation, to generate new candidate solutions. Each genetic operator has a probability of being applied to an individual. If none of the probabilities are met, the individual is copied to the next generation.

5. Population replacement: A new population is created using the offspring generated by the genetic operators and the original population based on some replacement criteria.

6. Termination Criteria: The process is repeated iteratively until a stopping criterion is met, such as a maximum number of generations or a target fitness value.

This process is not very different from the natural selection process, where individuals with favorable traits are more likely to survive and reproduce, passing their genes to the next generation. Over time, the population evolves towards better solutions, guided by the principles of natural selection. A visualization of the evolutionary algorithm process is shown in Fig. 2.3.

2.3.1 Genetic Programming

Genetic Programming (Koza, 1992) was later introduced as a new branch of evolutionary computation in the 1990's. It is a type of evolutionary algorithm that evolves computer programs to solve complex problems. Genetic programming is based on the

principles of natural selection and Genetic Algorithm (GA) and evolves a population of computer programs with variable lengths and complex structures.

In GA, the individuals are represented as fixed-length strings of binary or real-valued numbers (Vanneschi & Silva, 2023). This can be a limitation when dealing with problems that require variable-length solutions, such as symbolic regression, where the goal is to find a mathematical expression that fits the data. Genetic Programming (GP) overcomes this limitation by representing individuals as computer programs, which can have variable lengths and complex structures (Koza, 1992).

GP is a method for automatically generate computer programs that solve a problem. Data models are particular cases of computer programs, and GP can be used to evolve mathematical expressions that fit the data. Therefore, GP can be seen as a Machine Learning method, as it can be used to learn from data and make predictions.

Being a specific type of evolutionary algorithm, GP follows the same general strategy presented in Fig. 2.3, inspired by the theory of evolution. However, the representation of individuals, the genetic operators, and the fitness evaluation process are specific to the problem being solved.

In the following sections, the main components of a GP system will be presented.

2.3.1.1 Representation of Individuals

In GP, individuals are represented as computer programs, which can have variable lengths and complex structures. There are several ways to represent individuals, including linear (Brameier & Banzhaf, 2007), grammar-based (Whigham, 1999), graph-based (Sotto et al., 2021), and cartesian (J. F. Miller & Thomson, 2000) representations, in addition to the original and widely used tree-based representation.

The tree-based representation, proposed by Koza, 1992 and still widely used today, represents individuals as syntax trees. In these trees, the nodes represent functions or operators, and the leaves represent terminals or constants. The functions set, represented by $F = f_1, f_2, \ldots, f_n$, specifies the primitive functions. Each function is characterized by an arity, which is the number of arguments it takes. Unary function, such as sin and log take one argument, while binary functions, such as $+$ and $*$, take two. The terminal set, represented by $T = t_1, t_2, \ldots, t_m$, specifies the terminals, which can be variables of the input data or constants. The function and terminal sets are used to build the program trees - the individuals in the population (Vanneschi & Silva, 2023).

In Symbolic Regression (SR), the goal is to find a mathematical expression that fits the data. Therefore, the individuals in the GP population are mathematical expressions represented as trees, where the nodes represent mathematical functions, such as addition, subtraction, multiplication, and division, (ex.: $F = \{+, -, *, /, sin, exp, \ldots\}$) and the leaves represent constants or variables (ex.: $T = \{x_1, x_2, \ldots, x_p, 1, 2, e, \ldots\}$). The trees are constructed by recursively applying the functions and terminals, starting from the leaf nodes until the root node is reached. The combination of functions and terminals

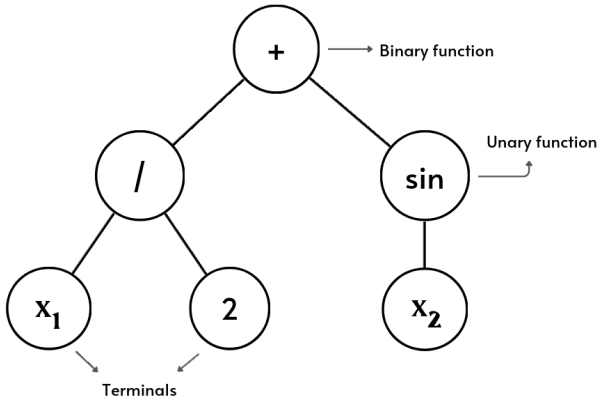


Figure 2.4: Tree-based representation of an individual in GP

might seem infinite, creating an infinite number of possible solutions. However, the search space can be limited by defining a maximum depth of the tree (Poli et al., 2008).

The structure of the function is also known as its *genotype*, as it contains the information that encodes the individual. A visualization of this tree-based representation is shown in Fig. 2.4, which encodes the mathematical expression $f(x) = (x_1/2) + sin(x_2)$

This tree structure allows for the representation of complex relationships and interactions between variables, making it well-suited for symbolic regression and other problems that require variable-length solutions.

2.3.1.2 Population Initialization

The initialization of the population is the first step in the GP process. It usually consists of generating random individuals that represent candidate solutions to the problem and that will later be evolved and improved with the goal of finding the best solution. Usually, the initial individuals have a pre-defined maximum depth d, which limits the size of the trees.

There are several initialization methods, but the most common ones are the *Grow* method, the *Full* method, and the *Ramped Half-and-Half* method (Koza, 1992).

Grow: This method generates trees of random depth, where each node has a probability of being a function or a terminal. Initially, a random function is selected from F as the root node. Then, depending on the arity of the function, symbols are randomly selected from $F \cup T$ to build the tree. The process is repeated until all leaves are filled, or the maximum depth is reached. This method is useful for creating diverse populations with different tree structures.

Full: This method generates trees of fixed depth, where all leaves are at the same level d. It works similarly to *grow*, with the difference of, instead of choosing a random symbol from $F \cup T$ at each step, the method selects a function from F as the root node

and recursively builds the tree until the maximum depth is reached. This method is useful for creating populations with uniform tree structures.

Ramped Half-and-Half: This method combines the *Grow* and *Full*. It was introduced to create diverse populations, composed by trees that are not too similar to each other. The method divides the population equally among individuals of all possible depths, ranging from 1 to d. For each depth, half of the individuals are generated using the *Grow* method, and the other half using *Full*. This method is useful for creating diverse populations with a mix of tree structures. The pseudo-code for this initialization process is shown in Algorithm 1.

Algorithm 1: Ramped Half-and-Half Initialization

```
Data: Population size S, Maximum depth d
Result: Population P
inds_per_depth ← S/d;
for max_depth ← 1 to d do
    for j ← 1 to inds_per_depth/2 do
        ind_grow ← grow(max_depth);
        ind_full ← full(max_depth);
        P ← P ∪ {ind_grow, ind_full};
    end
end
return P;
```

2.3.1.3 Fitness Evaluation

The fitness evaluation is a crucial step in the GP process, as it determines how well each individual performs on the optimization problem. It is used to guide the search towards better solutions by selecting the fittest individuals for reproduction (Koza, 1992). Fitness is also known as *phenotype*, as it represents the expression of the individual's *genotype* in the environment, which is the optimization problem being solved.

In SR, the fitness of an individual is typically evaluated by comparing its output to the target values in the training data. The fitness function is a measure of how well the individual fits the data and can be defined in different ways, depending on the problem being solved. As detailed in Section 2.1.2, in regression problems, where the goal is to find a mathematical expression that maps the inputs to the continuous outputs, the fitness function can be defined as, for example, the RMSE or MAE between the predicted values and the target values. On the other hand, in classification problems, where the goal is to classify instances into different categories, the fitness function can be defined as the accuracy, precision, recall, F1 score, or AUC-ROC.

2.3.1.4 Selection Mechanisms

The selection mechanism is responsible for choosing the individuals that will be used to generate the offspring in the next generation. This process is based on the fitness of the individuals, with fitter individuals being more likely to be selected (Koza, 1992). This approach aligns with the principles of natural selection, where individuals with favorable traits are more likely to survive and reproduce. By selecting the fittest individuals, the population evolves towards better solutions over time.

The most common selection mechanisms are the tournament selection, roulette wheel selection, and rank-based selection. Below is a brief explanation of the tournament selection mechanism, which is widely used in GP.

Tournament Selection: In Tournament Selection (B. L. Miller & Goldberg, 1995), a subset of individuals of size k is randomly selected from the population, and the fittest individual in the subset is chosen as the winner. The value of k is a hyperparameter that determines the selection pressure, with higher values leading to stronger selection pressure. This happens because, the higher the value of k, the higher the probability of including the fittest individual in the tournament, therefore, selecting it. It is possible to select the same individual multiple times to participate in the tournament, even if that individual is the worst in the population. This is a feature of Tournament Selection that allows for exploration of the search space, as it introduces randomness in the selection process. The pseudo-code for the tournament selection is shown in Algorithm 2, assuming the goal is to minimize the objective function.

Tournament selection is a simple and effective selection mechanism that ensures diversity in the population. It also avoids the evaluation of all individuals in the population, which can be computationally expensive.

Algorithm 2: Tournament Selection

Data: Population pop, Tournament size S, Objective $objective$
Result: Tournament winner $winner$
$participants \leftarrow \{\}$
for $i \leftarrow 1$ **to** S **do**
    $random\_ind \leftarrow$ Randomly select an individual from pop;
    $participants \leftarrow participants \cup random\_ind$;
end
$winner \leftarrow$ Individual from $participants$ that minimizes $objective$;
return $winner$;

2.3.1.5 Genetic Operators

Genetic operators are used to generate new candidate solutions by combining the genetic material of the selected individuals. They are variational operators, as they

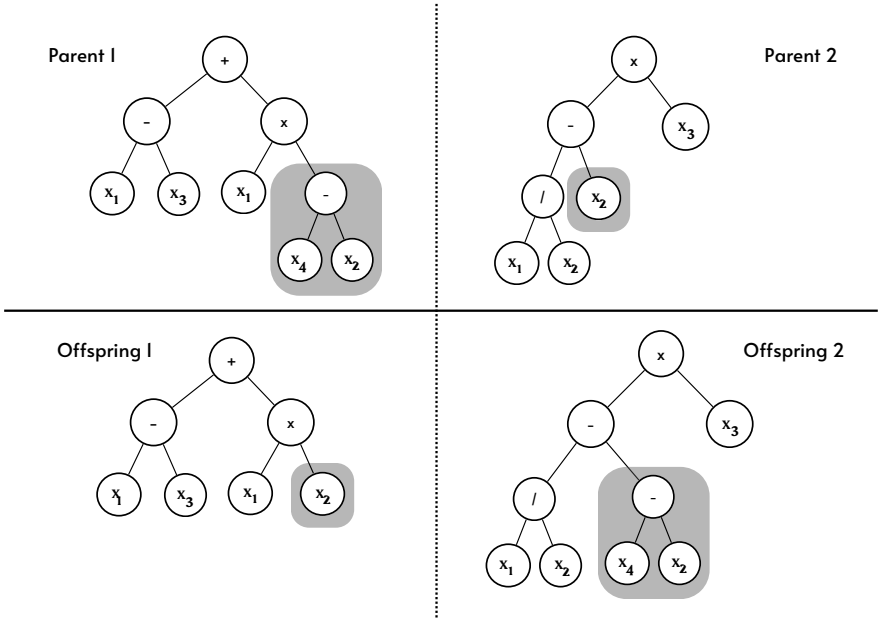


Figure 2.5: Standard subtree crossover operator in Genetic Programming

introduce variation in the population by creating new individuals that are different from their parents. The main genetic operators used in GP are crossover and mutation.

When the genetic operators were initially introduced (Goldberg, 1988; Holland, 1992), they were formulated for genetic algorithms, where individuals are represented fixed-length strings of binary or real-valued numbers. However, since GP individuals are represented as trees, Koza, 1992 introduced adaptations of these operators to work with tree-based representations.

Crossover: The standard subtree GP crossover operator works by exchanging subtrees between two parent individuals to create two offspring trees. The crossover point is randomly selected in each parent tree, and the subtrees below the crossover point are swapped between the parents (Poli et al., 2008). This process creates two new individuals that combine genetic material from both parents. A visualization of this process is shown in Fig. 2.5.

The crossover operator always produces syntactically legal offspring, as the subtrees are swapped at the subtree root node level. This variational operator is used to explore the search space and combine different genetic material to generate new candidate solutions.

Mutation: The standard subtree mutation operator works by randomly changing a subtree in an individual to create a new individual. The mutation point is randomly selected in the individual, and the subtree below the mutation point is replaced by a new subtree (Poli et al., 2008). This operation complies with the maximum tree depth defined for the GP evolution. The mutation process introduces random changes in the

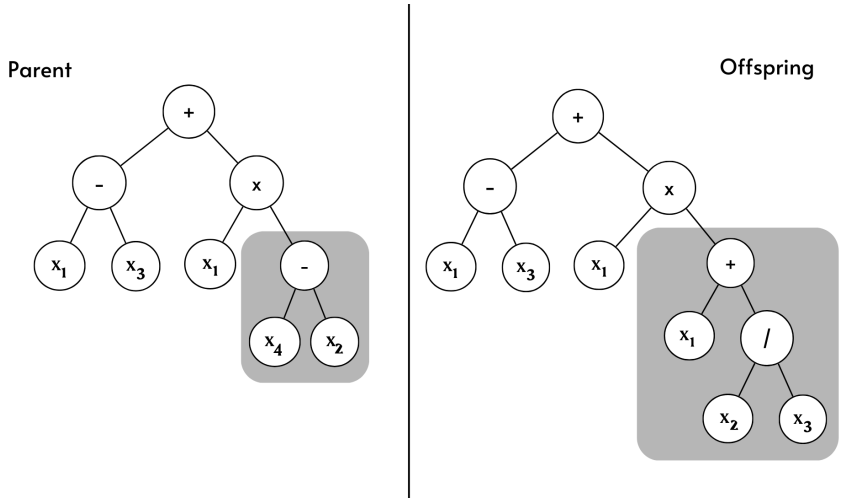


Figure 2.6: Standard subtree mutation operator in Genetic Programming

population and helps explore new regions of the search space. A visualization of this process is shown in Fig. 2.6.

3

Literature Review

3.1 Smart Data Usage Methods

The challenge of overfitting has promoted innovative approaches to identify and mitigate its impact. The implementation of strategies that use data intelligently has seen a growing interest, resulting in the proposal of several solutions in this context.

When it comes to avoiding overfitting, the initial challenge is being able to identify it. Foreman and Evett (2005) uses *canary functions* (Evett et al., 1998) as an early warning system. These functions are designed to be similar to the main fitness function but start diverging from it when overfitting occurs. By monitoring these functions, a consistent degradation can serve as an indicator of overfitting. The authors used the fitness of the best-of-run individual in a separate validation set as *canary function*. This approach aligns with a broader trend in the Machine Learning community, where the use of dedicated validation sets for detecting overfitting has become increasingly prevalent - a method now applied to genetic programming in this study.

Beyond simply identifying overfitting, a validation set also serves as a practical tool for preventing it. Robilliard and Fonlupt (2002) propose *Backwarding*, a method that evaluates the new best model on both the training and validation sets and selectively stores solutions that improve in both, ensuring the return of a non-overfitted solution at the end. More recently, Daniel Rivero and Pazos (2020) takes the *Backwarding* approach a step further, considering multiple best individuals in each generation instead of the single best-of-run. This broader approach takes into account cases where other individuals, with higher training errors but lower validation errors, might be less overfitted. The best validation individual, not necessarily the training best in any generation, is chosen at the end.

In addition to validation sets, the strategic partitioning of the training data itself has been proposed as a technique to make GP models generalize better. The idea was introduced by Gathercole and Ross (1994) as the *Random Subset Selection* technique, initially designed to accelerate GP runs. It was later refined and renamed to *Random Sampling Technique* (RST) in Gonçalves et al. (2012), where the authors adapt it for

overfitting avoidance. This newly proposed strategy promotes generalization by evaluating the individuals on a random subset of the training data in each generation. As a consequence, the process ensures that only individuals that are good on diverse subsets are retained in the population, ultimately improving their ability to generalize. Another noteworthy approach is *Interleaved Sampling*, introduced by the same authors in Gonçalves and Silva (2013). This method strategically alternates between using a single training instance and all instances, aiming to find a balance between minimizing overfitting (with a single instance) and capturing general patterns (using the entire dataset).

Finally, a technique called *Bootstrapping*, inspired by the Bias-Variance decomposition, has proven to be another valuable method. The Bias-Variance decomposition breaks down the error of a predictive model into two components: bias, representing the model's systematic error or tendency to oversimplify, and variance, reflecting its sensitivity to changes in the training data, which indicates the model's ability to generalize. Several works, including Agapitos et al. (2012) and Fitzgerald et al. (2013), propose a technique to evaluate the sensitivity of evolved models to a specific training dataset using boostraping. The approach involves randomly drawing datasets with replacement from the original training and using the variance of the error on these bootstrap samples, also knows as *Boostrap Standard Error* (BSE), along with the error on the original training dataset, as a single-objective fitness function. In the former work, the authors use a weighted sum as fitness function, with coefficients representing the trade-off between bias and variance, while in the latter, fitness is calculated by multiplying the percentage error rate on the training data by the BSE.

3.2 Ensemble Methods

Ensemble methods are another powerful approach extensively studied by the Machine Learning community. This technique involves combining predictions from multiple models to improve overall performance, aiming to mitigate the limitations of individual models by leveraging the strengths of several diverse ones. The fundamental idea is that, by taking advantage of predictions from different models, the ensemble can achieve better generalization and predictive accuracy compared to any single model. Individual models within an ensemble often specialize in capturing different and specific patterns of the underlying data, either due to using different training subsets, different model architectures or hyperparameters. When these specialized models are combined, their collective prediction becomes a robust and more generalized representation of the underlying patterns in the data.

Keijzer and Babovic (2000) provides an introductory analysis of the Bias-Variance decomposition in GP, both theoretically and experimentally, highlighting the potential of ensemble methods in addressing overfitting. Specifically, the authors explain that

while *bagging* averages predictions from multiple independently trained models, reducing variance, *boosting* focuses on minimizing bias by combining weak models into a strong one, sequentially. A practical implementation of these conceptual findings can be found in the work of Paris et al. (2004), where the results show that *boosting* is able to filter noise effectively in regression problems, exhibiting an error on the noise-free test set that is smaller than the error on the noisy learning set. This suggests that, despite the inherent structural complexity of GP, boosting succeeds in mitigating overfitting effects.

Moreover, a recent study by Castelli et al. (2018) introduces an innovative approach to building GP *bagging* ensembles by blending individuals from *Standard Syntax-Based GP* (STGP) and *Geometric Semantic Genetic Programming* (GSGP). The proposed ensemble-building approach involves evolving parallel GP populations, with some employing STGP and others GSGP. The ensemble's final value is computed as a weighted sum of the semantics of the best individuals, with weights adjusted through a pruning process based on semantic similarity. The pruning ensures diversity among the models, contributing to improved overall generalization performance, as demonstrated in experiments across various benchmarks.

3.3 Complexity Minimization Methods

3.3.1 Structural Complexity

In the early exploration of overfitting in Genetic Programming, a significant focus was placed on the association between overfitting and the phenomenon of bloat - an excess of code growth without a corresponding improvement in fitness. This connection aligns with the principles of Occam's Razor and the Minimum Description Length (MDL).

The Occam's Razor principle, a well-known philosophical hypothesis, states that simpler models are preferable if they can achieve comparable predictive performance to more complex ones. MDL, on the other hand, provides a practical framework for model selection based on the belief that the smaller the code needed to encode a model, the less complex it is. This notion is referred to as structural complexity, establishing a connection between representation size (i.e., genotype) and model complexity. In the context of GP, this translates to the objective of evolving models that are as concise as possible while still achieving good predictive performance, avoiding overly complex models that may not generalize well to new, unseen data.

Early studies, such as the pioneering work by Iba et al. (1994), delved into the relationship between the MDL principle and GP performance. The study revolves around the belief that Kolmogorov complexity - the length of the shortest computer program producing a given object - is a good measure of model complexity. However, due to the uncomputability of Kolmogorov complexity, an approximation is used. This

approximation involves calculating the length of code required to encode a program (a function of the tree size) and its predictive error (a function of the errors). Using this approximation, the study introduces an MDL-based fitness function to control the growth of individuals. It is important to mention that the study relies on a decision tree representation instead of the standard tree representation in GP, leveraging specific properties that make MDL a reliable approach. For instance, the decision tree representation aligns with the notion that, the more the tree grows, the fitter it becomes, which is usually not the case with standard GP trees. Therefore, the authors acknowledged that the successful application of this MDL-based fitness function for overfitting control requires careful consideration due to its dependence on the properties of the tree's representation.

Subsequent research explored the connection between limiting bloat and improving generalization. In Zhang and Mühlenbein (1995), the authors focused on evolving sigma-pi neural networks, introducing a size penalty to regulate fitness and improve generalization. An adaptive parameter dynamically adjusted during evolution guarantees a balance between error reduction and complexity, resulting in more parsimonious solutions. Similarly, Rosca (1996) studies the relationship between bloat and generalization in the context of the Pac-Man game, where individuals are represented as a set of the game's instructions. The analysis suggests that monitoring size in GP can offer insights into the search space properties and improve the understanding of the dynamics of evolution in terms of generality, and modularity.

Additionally, The Tarpeian Bloat Control method, introduced by Poli (2003), addresses bloat by imposing penalties on larger individuals, promoting the evolution of smaller and more parsimonious solutions. However, as investigated by Mahler et al. (2005), the impact of this method on generalization in Genetic Programming varies across different problems. Their experiments in symbolic regression reveal a problem-dependent effectiveness: while it improves generalization accuracy in some cases, it may worsen it in others, particularly with higher Tarpeian Control ratios.

Another approach to control bloat in GP is the Lexicographic Parsimony Pressure method, introduced by Luke and Panait (2002b), which involves an additional criterion during the evaluation of individuals. When multiple solutions exhibit similar primary fitness, the algorithm considers the secondary criterion of simplicity or size. In a study conducted by Gagné et al. (2006), investigating the impact of this bloat control method on GP for binary classification, experimental results across six datasets indicate a significant reduction in the size of the best-of-run individuals without sacrificing accuracy on test sets. However, it is important to note that this method, on its own, did not result in an improvement in generalization.

In a later work, Kronberger et al. (2011) introduces an adaptive approach for controlling parsimony pressure, aiming to mitigate overfitting in symbolic regression more intelligently. The method dynamically adjusts the desired average program length based on the detection of overfitting, by monitoring the correlation between the

best-of-run training and validation fitness values. The conclusion suggests that, while the method shows promise, further research is needed to address challenges, such as the relationship between program length and complexity, testing on additional datasets, and fine-tuning parameters for practical applications.

Despite being a long-standing assumption, with several studies claiming an association, some research has questioned the effectiveness of bloat control methods in improving the generalization ability of GP. Particularly, Silva and Vanneschi (2009), Silva et al. (2012) and Vanneschi et al. (2010) have demonstrated that, even when applying one of the most effective bloat control methods to a GP run, specifically Operator Equalisation (Dignum & Poli, 2008), the system remains free of bloat but still overfits. This suggests that, contrary to the assumption, relying on size alone as a proxy for complexity may not be sufficient to control overfitting.

3.3.2 Functional Complexity

Given the limitations of using size as a unique indicator of complexity, recent studies have turned their attention to the concept of functional complexity. Contrary to structural complexity, which is based on the individuals' genotype, functional complexity is based on the behavior of the individuals, i.e. their phenotype.

One of the initial studies following this new approach (Vladislavleva et al., 2009) introduces a complexity measure called the order of nonlinearity, calculated based on the minimum degree of the Chebyshev polynomial approximating the function surface. While proposing a new functional complexity measure, the study still considers structural complexity as another objective to minimize. The experimental results show that alternating optimization objectives, namely the error and structural complexity, and the error and functional complexity, leads to solutions that are both compact and possess smoother response surfaces, contributing to better interpretability. However, there are some limitations in the approach, such as the potential overestimation of functional complexity, and the challenge of extending it to multivariate functions.

Other studies explored the concept of Tikhonov Regularization in GP, also known as Ridge or L2 regularization, which involves adding a regularization term to the optimization objective. In GP, this regularization term is represented by a linear combination of derivatives of a function, penalizing the complexity of individuals and promoting smoothness. For instance, when incorporating second-order partial derivatives into the Tikhonov regularization term, the functions are penalized for excessive "wiggliness", suppressing overfitting by discouraging rapid changes. In the work by Wu et al. (2006), the authors integrate this regularization into the fitness function, dynamically adjusting the degree of penalization. This adaptive approach initially focuses only on minimizing the error until a predefined threshold is surpassed, at which point the fitness function begins penalizing complexity by incorporating the regularization term. This adaptability ensures that, only towards the end of a GP run,

both training error and complexity are considered in the fitness function, encouraging diversity early on and promoting smoother models with better generalization in later stages. On the other hand, a study by Ni and Rockett (2015) extends the application of Tikhonov regularization as a functional complexity measure, this time within a multi-objective setting, eliminating the need to manually select the regularization degree parameter.

Another relevant study (Azad & Ryan, 2011) explores the smoothness of response surfaces of GP models by calculating the variance of the output over the training data. Throughout evolution, both error and variance are minimized using a modified tournament scheme, facilitating a single-objective framework that optimizes two objectives concurrently, eliminating the need for multi-objective optimization. This scheme first decides between two candidate solutions by establishing Pareto Dominance. If neither model dominates the other, the evaluation considers whether one model improves over the other in one objective without significant compromise in the other. To decide this, the rectilinear distance of one solution ($|error + variance|$) is compared with the Euclidean distance of the other solution ($\sqrt{error^2 + variance^2}$). However, if it is still not possible to make a decision, the model with smaller variance is chosen. This scheme is particularly elegant as it consolidates two objectives into a single-objective framework, eliminating the complexity of choosing an individual from the Pareto front in a multi-objective setting. However, using variance as a proxy for complexity has limitations, such as its unawareness of changes over the input axis and the fact that a high variance over a large input domain may still denote a smooth surface. Therefore, variance might be a misleading indicator of complexity in some cases.

Additionally, the strategy proposed by Mousavi Astarabadi and Ebadzadeh (2015) suggests improving the generalization capability of GP by regulating the first-order derivative of GP trees throughout the evolutionary process. This involves employing a multi-objective GP approach, where, in addition to the conventional RMSE as the primary minimization goal, the complexity of GP trees is assessed using the RMSE between the first-order derivatives of the desired solution and the GP tree. Despite the unknown nature of the target function and its derivatives, estimates can be derived through limit calculations over the training points. For multivariate functions, the directional derivative in the direction of the nearest point is estimated at each training data point. The function set is predefined, and the derivative rules for each function are derived à priori, ensuring computational efficiency and feasibility in individual's fitness calculation. Although the observed differences in test performance were supportive of the method's effectiveness, they were only reported in the last generation of the experiments and did not reach high significance. This suggests that, while the proposed approach shows promise, further investigation may be necessary to fully determine its impact and potential limitations.

A study by Vanneschi et al. (2010) introduces a new complexity measure inspired

by the concept of curvature (Morvan, 2008), considering it as an indicator of complexity. The curvature of a function represents how much its geometric representation deviates from being flat or straight. Although the formal computation of curvature is challenging, requiring the calculation of first and second derivatives over each dimension, the proposed measure provides an approximation of the average curvature of a GP individual, without needing to compute derivatives. This approximation is achieved by evaluating the slopes of segments in its graphical representation across each dimension (feature). The intuition behind the measure involves summing the values of the difference between consecutive slopes, assigning higher weights to inversions in slope signs and reflecting the complexity of the function. This approach provides a computationally efficient means to assess functional complexity in polynomial time with the number of fitness cases, offering valuable insights into the evolving complexity of GP individuals during the evolutionary process.

Similarly, Trujillo et al. (2011) propose Holderian complexity as a measure of the regularity or irregularity of functions, reflecting their curvature. The Holder exponent quantifies the point-wise regularity of a signal at each point of the function, aiding in identifying irregularities or singularities indicative of points of non-differentiability. The resulting Regularity-based Functional Complexity (RFC) measure offers insights into the behavior of complex functions. Moreover, Castelli et al. (2011) propose Graph-Based Complexity (GBC), aligning with the notion that complex functions exhibit a substantial *ruggedness* in their behavior. The calculation involves assessing complexity by examining pairs of *close* training points and the distance between their outputs, where the term *close* is determined by a specified distance threshold δ in the input space. The GBC measure quantifies the proportion of training points where the function is rugged, providing a measure of its curvature. This study also introduces the concept of Graph-Based Learning Ability (GBLA), an extension of GBC, evaluating the GP individual's capacity to learn what are called *difficult* points, where the learned and target functions exhibit differences in *ruggedness*. These measures are used in a novel fitness function, designed to balance fitness (RMSE) and complexity measures (GBC and GBLA). This fitness function demonstrates improved generalization ability in GP compared to standard approaches.

Moreover, a recent study by Vanneschi and Castelli (2021) improves the GBC proposal, introducing a new measure of functional complexity called Input/Output Distance Correlation (IODC). In contrast to the original Graph-Based Complexity, IODC assesses the correlation between pairwise distances of input data and corresponding distances of output values. The distinctive feature of IODC is its parameter-free nature, eliminating challenges associated with parameter tuning in GBC. IODC cleverly measures models curvature by calculating the correlation between pair-wise distances of observations and pair-wise distances of their outputs. The study demonstrates that optimizing IODC alongside RMSE using nested tournaments, while incorporating SoftTarget regularization in a hybrid approach, surpasses the generalization ability of

traditional GP methods.

Finally, Adaptive Weighted Splines (AWS), as introduced in Raymond et al. (2020), challenges the traditional tree-based representation in Genetic Programming by offering a distinctive advantage. The proposed representation allows for an explicit control over both structural and functional complexity, addressing the challenges faced by tree-based GP in quantifying model complexity directly. AWS uses smoothing splines, a specific type of splines, which are piece-wise polynomial functions minimizing fitting error and a smoothing penalty. In AWS, each individual consists of feature splines, composed by a smoothing spline, primary coefficients (for feature selection), and secondary coefficients (for feature weighting). These components represent the characteristics of an individual and are all optimized during evolution. The use of smoothing splines ensures a smooth representation, beneficial for generalization purposes. AWS shows a substantial improvement in generalization performance across various benchmark datasets, consistently outperforming Standard GP in terms of both training and testing performances. However, it comes with trade-offs, giving up certain symbolic properties and imposing a fixed model structure, which could limit interpretability and pose challenges to capturing complex relationships. The explicit feature control, increased representation complexity, and dependence on initialization could present difficulties in specific scenarios.

3.3.3 Statistical Methods

Approaches inspired in the statistical learning theory have also been proposed. At the core of this theory is the concept of a hypothesis class, representing a set of potential models or functions that a learning algorithm can choose from. The primary objective of the learning algorithm is to identify the optimal model from this hypothesis class, one that not only fits the training data well but also maintains a balance by avoiding excessive complexity, ensuring generalization with unseen data.

In the context of Genetic Programming, statistical learning theory has been applied to derive generalization bounds that quantify the expected difference between the performance of a model on the training data and its performance on unseen data. A key principle in this context is Structural Risk Minimization (SRM), which provides a framework to estimate the difference between generalization error and empirical error. The Vapnik-Chervonenkis dimension (VC-dimension) (Vapnik, 1999) plays a crucial role in this estimation, serving as a metric to quantify the capacity or complexity of a class of models. It is defined as the size of the largest set of points that the class can shatter - a set is considered shattered if, for every possible binary labeling of the points, there exists at least one function in the class capable of achieving that particular labeling. Despite SRM's solid theoretical background, it has seen limited application in GP due to challenges in measuring the VC dimension of nonlinear models.

Chen et al. (2016) addresses this challenge by proposing an empirical approach to

approximate the VC-dimension in GP. The method involves measuring the empirical maximum deviation of a model on two independent datasets of the same size. This deviation is defined as the difference in error frequencies between the two datasets. The calculated VC-dimension is then integrated into the SRM framework, which is applied to GP through the fitness function. The fitness function, guided by the VC-dimension bound, aims to find a good trade-off between empirical error and model complexity. While the approach has shown success in improving generalization over standard GP, the paper acknowledges certain drawbacks. Specifically, it highlights the use of a uniform setting used to measure VC-dimension, potentially affecting the accuracy of the estimated generalization error. This arises from running the same number of experiments to determine maximum deviation errors across all datasets, regardless of the number of instances. In response to these limitations, Chen et al. (2019) propose an improved algorithm that employs a nonuniform setting for measuring the VC-dimension. This new setting is expected to provide a tighter VC generalization bound, crucial for the success of SRM-driven GP. This advancement is designed to decrease the likelihood of selecting overly complex models during breeding, reducing over-adaptation to training data and promoting better generalization on unseen data.

Another innovative approach to face the challenge of computing the VC-dimension is proposed in Alonso et al. (2016). In contrast to the conventional tree-like structures used for model representation in GP, the paper introduces the use of straight line programs (SLP). Calculating the VC-dimension for tree structures is challenging because the mathematical analysis becomes intricate due to the complex branching nature of trees, making it difficult to derive a clear and computationally feasible measure. On the other hand, the SLP representation has a more sequential and linear structure, making it easier to analyze mathematically and calculate the VC-dimension. Similarly to what was done in previous work, the authors evolve the GP models using a fitness function that incorporates this measure of model complexity. Nevertheless, the SLP representation comes with trade-offs, making it challenging to smoothly apply to every GP application, as it diverges from the standard practices.

Lastly, another application from statistical learning theory is explored in Chen et al. (2022), where the authors employ Rademacher complexity to guide the evolution process. Rademacher complexity serves as a measure of the capacity of a set of models to fit random variables, calculated as the maximum correlation between a model and Rademacher variables on selected training instances. This data-dependent metric is advantageous for providing tighter generalization bounds than traditional measures like VC-dimension, making it a valuable tool in estimating the behavioral complexity of GP models. In the paper, the computation of Rademacher complexity involves generating Rademacher variables (random variables with values of +1 or -1) and determining the correlation between the model's outputs and these variables. An adaptation is made for regression models, introducing the hyperbolic tangent (tanh) transformation to map continuous outputs into the required range [-1, 1]. The method

proposed in the paper incorporates Rademacher complexity as the second objective alongside minimizing training error in a multi-objective GP setting.

3.4 Other Methods

Over the years, different and varied methods have been proposed to address the challenge of overfitting in Genetic Programming. These methods don't fit into any of the previous categories, but still deserve to be mentioned.

For instance, Banzhaf et al. (1996) investigates the impact of an aggressive use of the mutation operator on GP's generalization performance. In contrast to the typical reliance on crossover, the study employs multiple mutation rates and the results show that higher rates significantly improve generalization performance. This improvement is particularly evident on challenging datasets, suggesting that increased diversity in the population plays a central role. Moreover, changes in training indicators, such as a reduction in introns and an increase in effective size, indicate that the mutation operator helps maintain diversity and evolvability, successfully avoiding premature convergence issues.

Furthermore, the approach proposed by Da Costa and Landry (2006) suggests relaxing the training set in GP by widening the definition of the desired target solution. Results show that slight relaxation enhances generalization error, indicating improved performance. Additionally, G. Murphy and Ryan (2008) introduces Hereditary Repulsion (HR), a convergence manipulation algorithm, whose key innovation lies in a simple constraint - individuals must outperform both parents to be considered for the next generation. This constraint prevents premature convergence and overfitting by promoting recombination events that combine useful genetic material from both parents.

In addition, Vanneschi and Gustafson (2009) propose Genetic Programming with Repulsors (repGP), aiming to identify individuals overfitting the training set and treat them as *repulsors*. These, along with their structurally similar counterparts, are excluded from the evolutionary process to enhance generalization. The methodology involves maintaining a list of *repulsors* and applying a two-layer tournament selection algorithm. If an individual overfits the training set, it is added to the *repulsors* list. During selection, individuals similar to *repulsors* are excluded. The paper explores two variants of repGP: ED-repGP using structural distance to calculate the similarity between individuals and *repulsors* and SCD-repGP using subtree crossover-based similarity measure. Both variants demonstrate improved generalization compared to standard GP.

Moreover, recent innovative works have been published, such as Chen et al. (2017), where the authors propose GP with Permutation Importance (GPPI) to enhance GP's generalization for high-dimensional symbolic regression. GPPI collects features from

the best-of-run individuals across multiple GP runs and employs permutation importance, a measure widely used in Random Forest, to quantify the importance of each feature by evaluating the impact of permuting its values on the model's generalization error. Features with positive permutation importance, indicating a positive effect in reducing regression error, are selected for subsequent GP regression. Another novel approach, Asynchronous Parallel Genetic Programming (APGP) (Sambo et al., 2021), enhances GP efficiency by evaluating individuals asynchronously, allowing faster-evaluating individuals to enter the population ahead of their more complex counterparts. The algorithm uses a race condition, where less complex individuals evaluated quickly have the opportunity to replace their slower counterparts in the population. While speed provides an advantage, the decision to include an individual in the population is solely based on accuracy, ensuring that good models are not excluded.

Finally, a recent study by Bomarito et al. (2022) introduces a Fractional Bayes Factor fitness measure. This approach achieves significant generalization performance improvements in GP by integrating two key methodologies - Normalized Marginal Likelihood (NML) and Probabilistic Crowding (PC). NML serves as a Bayesian fitness metric that dynamically assesses the fitness of models based on their likelihood given observed data, penalizing excessive complexity. This allows the GP algorithm to adapt to dataset complexity, acting as a form of regularization. On the other hand, PC introduces probabilistic selection based on fitness values, fostering diversity and preventing premature convergence in the evolutionary process. This combined approach ensures a more robust and adaptable GP evolution, showing improved generalization ability and reduced bloat in solutions.

4

Methodology

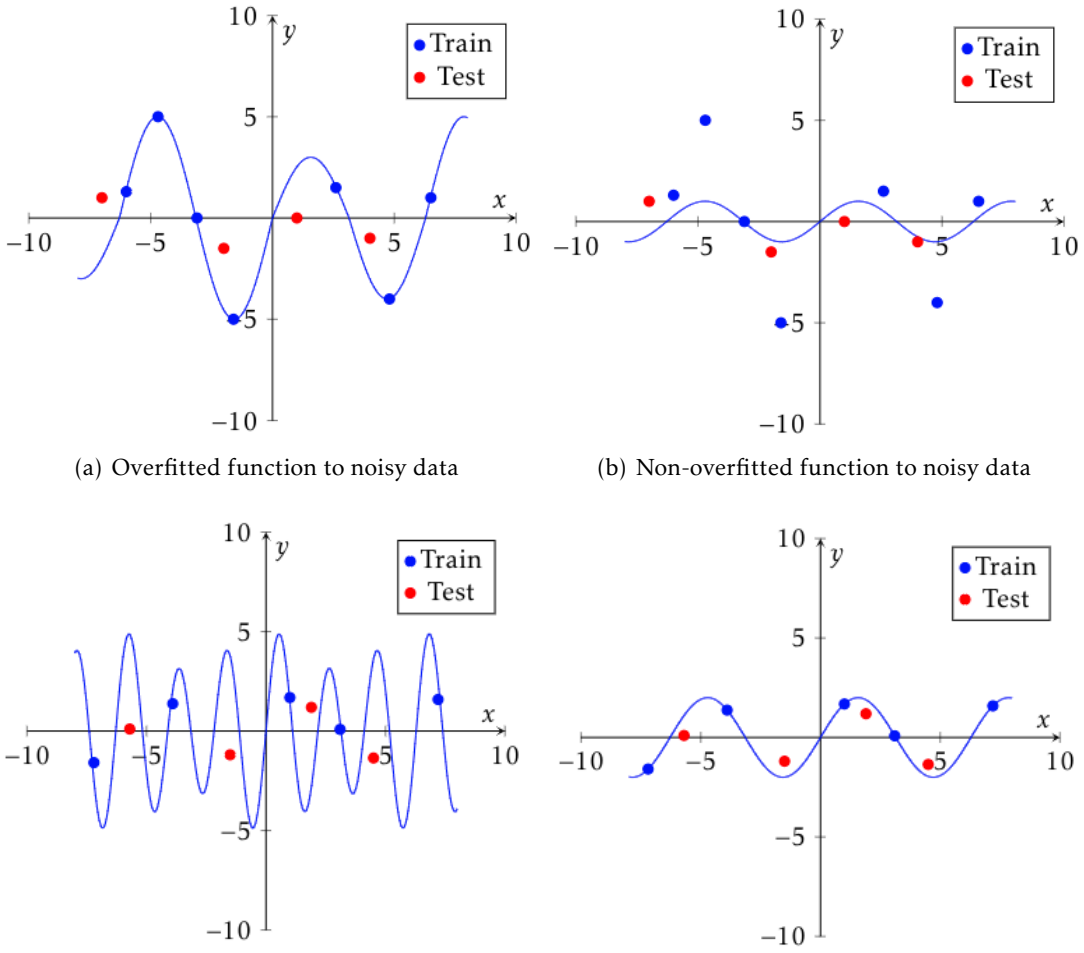
This chapter introduces the methodology employed in this study to address overfitting in Genetic Programming. This involves integrating a measure of functional complexity into the selection scheme during evolution. The intuition and motivation behind this approach are described in Section 4.1, followed by the definition of the functional complexity measure used, in Section 4.2. Additionally, Section 4.3 outlines a set of strategies for minimizing complexity alongside error throughout the GP evolution. Among these strategies, the Double Tournament selection and the Modified Multi-Objective Tournament Scheme are detailed on in Sections 4.3.1 and 4.3.2, respectively, as they were chosen for implementation and testing.

4.1 Motivation

Mitigating overfitting is crucial for ensuring the generalization of models beyond the training data. This study proposes a method that penalizes overly complex models during the selection step of the GP evolution, under the assumption that complexity often leads to overfitting. Central to this approach is the belief that the curvature of a function serves as a reliable indicator of its complexity.

To better understand the rationale behind the method, it is important to consider the relationship between functional complexity and generalization ability. Generally, a highly non-linear function will tend to closely fit all points on the training data, including noise, leading to poorer generalization. Take the functions in Fig. 4.1 as an example. The function shown in Fig. 4.1(a) accurately fits all the points in the training dataset, including the noisy ones, resulting in no training error. However, this function fails to generalize when applied to unseen data from the test set. On the other hand, the function in Fig. 4.1(b) is smoother and, despite having a higher training error, generalizes better to the test set.

Now consider Figures 4.1(c) and 4.1(d), both showing an identical set of training points, this time with residual to no noise. Given this set of training points, it is possible to find an infinite number of polynomials that perfectly fit them, with some



(a) Overfitted function to noisy data

(b) Non-overfitted function to noisy data

(c) Overfitted function to noiseless data

(d) Non-overfitted function to noiseless data

Figure 4.1: Functions with different generalization abilities.

prone to overfitting while others generalizing better. For instance, the function in Fig. 4.1(c) fits all the points in the training data, resulting in no training error, yet it shows high error when applied to the test set. On the other hand, the function in Fig. 4.1(d) also fits all training data points but demonstrates better generalization to the test set.

It is evident from the figures that the key difference between the overfitted and non-overfitted functions is their curvature. The overfitted functions show a much higher degree of curvature, with plenty of oscillations, while the non-overfitted functions are more linear. This observation highlights the intuitive link between curvature, complexity, and overfitting. Functions with a greater degree of curvature and plenty of oscillations often show a stronger tendency to perform poorly in terms of generalization. This intuition forms the basis for the proposed approach to mitigate overfitting in Genetic Programming.

To implement this approach, it is essential to create a method for approximating the curvature of a function. This measure will be used to evaluate the complexity of

models during evolution and penalize those that are overly complex, promoting the propagation of more generalized solutions.

In summary, the methodology involves two key steps: first, developing a reliable approximation for measuring the curvature of functions, and second, integrating this measure into the GP framework during the selection phase. This integration aims to guide the evolution towards simpler, less overfitted models that are still accurate. By leveraging the concept of curvature as a proxy for complexity, the approach proposed in this work seeks to improve the robustness and generalization abilities of evolved solutions.

4.2 Slope-Based Complexity

Measuring the complexity of a function is a challenge with no straightforward or widely accepted solution. Therefore, several measures have been proposed, as discussed in Chapter 3. However, the key assumption of this methodological approach is that the curvature of a function is a good indicator of its functional complexity, and that functional complexity, in turn, correlates with its tendency to overfitting. Thus, the complexity measure employed in this study is a phenotypical measure inspired by the concept of curvature. Unlike metrics that account for characteristics intrinsic to the function itself, such as the number of nodes or the length of the tree representation, this measure solely considers the output of the function.

Informally, the curvature of a function is the amount by which its geometric representation deviates from being *flat* or *straight* (Vanneschi et al., 2010). Morvan (2008) formally defined the curvature of a plain curve in 3 dimensions, given parametrically by $c(t) = (x(t), y(t))$, as:

$$k = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{\frac{3}{2}}} \tag{4.1}$$

As one can see, the curvature is determined by the first and second derivatives of the function in each dimension of the feature space. Because of that, calculating the exact curvature of each individual in a GP population would be prohibitively costly. Also, if the function representing an individual is non-differentiable, the exact calculation may be impossible. Moreover, if the feature space dimensionality is high, which is usually the case in Symbolic Regression problems, determining the curvature requires calculating even more derivatives, resulting in higher computational cost.

Therefore, Vanneschi et al. (2010) proposed a measure called *Slope-Based Complexity (SBC)*, which aims to provide a good approximation of the curvature of a function while being computationally efficient. SBC has the advantage of being computed in polynomial time with the number of fitness cases, and was designed to assign higher complexity values to functions showing high non-linearity. For instance, consider the

functions illustrated in Fig. 4.2. The idea behind SBC is to define a complexity measure that captures that the function in Fig. 4.2(c) is more complex than the one in Fig. 4.2(b), which is in turn more complex than the one in Fig. 4.2(a). Although all three functions consist of polylines, in Fig. 4.2(a), all segments forming the polyline have the same slope, whereas in Fig. 4.2(b), the segments have different slopes, despite all with the same sign. Finally, segments in Fig. 4.2(c) have varying slopes with different signs. Essentially, the measure aims to express the complexity of a function by accounting for the number of different slopes and assigning greater weight to changes in slope sign. It is important to mention that, although informally inspired by curvature, this measure has no formal relationship with the curvature measure.

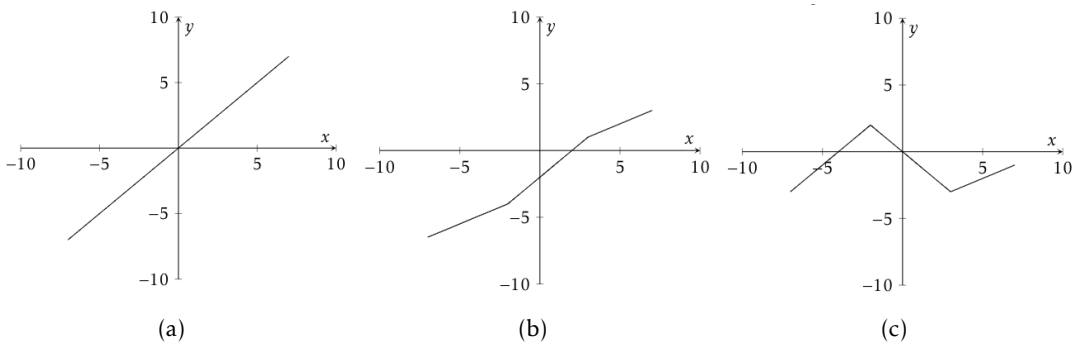


Figure 4.2: Three functions with different complexities.

The intuition of this indicator is very simple, although its formal definition may appear complex. Therefore, this measure is initially presented from an intuitive point of view before its formal definition is given.

Intuition: Imagine a GP individual as a polyline in a bi-dimensional space, with the points representing the fitness cases on the abscissas and the corresponding function values on the ordinates. The measure is calculated by sorting all fitness cases in ascending order and considering the values assumed by the GP individual on those fitness cases. Then, the slope of each segment joining these points is calculated. Let $s_1, s_2, s_3, s_4, s_5, \ldots$ be those slopes. The measure is simply calculated as:

$$|s1 - s2| + |s2 - s3| + |s3 - s4| + |s4 - s5| + \ldots$$

This calculation yields a value of zero if all slopes are identical, indicating minimal complexity. On the other hand, if consecutive segment slopes change signs, their contribution to the measure is maximal. When two consecutive segments have different slopes with the same sign, their contribution equals the absolute difference between their values, making it larger than zero, but smaller than in the case where the slope sign changes. The slope calculation is illustrated in Fig. 4.3.

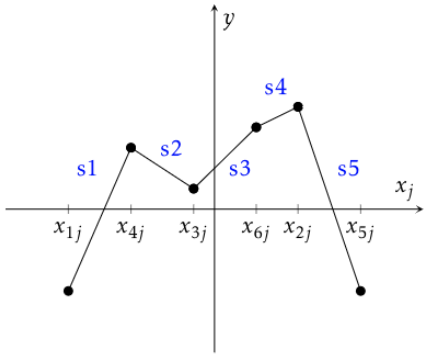
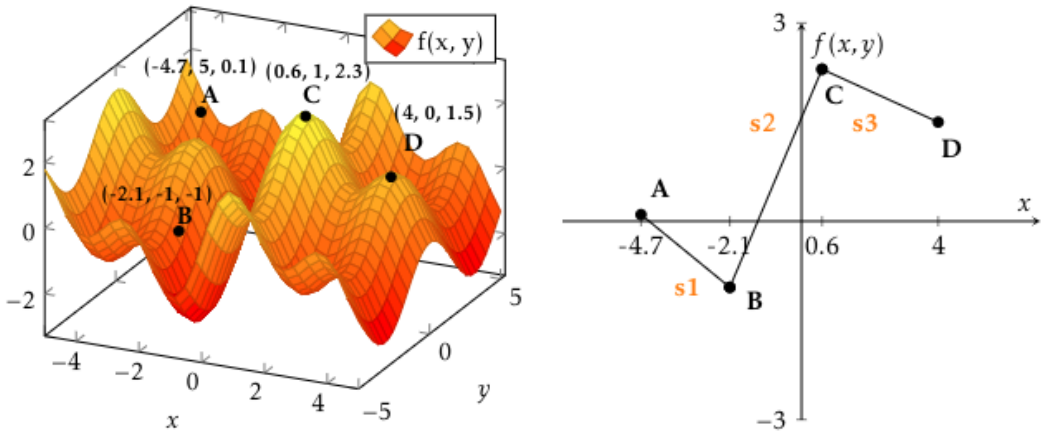


Figure 4.3: Slopes between each pair of consecutive fitness cases ordered by the values in feature j.



(a) Surface of a function representing an individual. (b) Partial complexity over the x dimension.

Figure 4.4: Practical example of the SBC partial complexity calculation.

In case of multidimensional spaces of features, the measure simply considers the projection of the fitness cases and corresponding outputs onto the $x_j z$ plane, where x_j is the j^{th} feature and z is the output of the function. The calculus is repeated for each dimension separately, and the results are aggregated using the sum, average, or another aggregation method.

For instance, consider the practical example in Fig. 4.4. Fig. 4.4(a) shows the surface of a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ coded by a GP individual. For each fitness case, the function value is calculated, and the points A, B, C and D are obtained. To calculate the partial complexity over dimension x, the points are projected onto the xz plane. This corresponds to a plane obtained by fixing all input dimensions with some value, except for the x dimension. Figure 4.4(b) shows this projection. In this new space, the slopes between consecutive observations are computed, and the partial complexity measure is calculated.

With the intuition explained, the formal definition of the *Slope-Based Complexity* measure can now be presented.

Formal definition: Let $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n$ be the fitness cases, where for each $i = 1, 2, \ldots, n : \mathbf{x}_i = (x_{i1}, x_{i3}, \ldots, x_{im})$ is a m-dimensional vector of floating point numbers. Let $g : \mathbb{R}^k \rightarrow \mathbb{R}$ be the function coding a GP individual that uses $k \leq m$ features, and let $g_i = g(x_i)$ be the output of g on the i^{th} fitness case. Given a $j = 1, 2, \ldots, k$, let $p_j = (x_{1j}, x_{2j}, \ldots, x_{nj})$ be a vector containing all the values of feature j in $\mathbf{X}$. Now let $q_j = (y_{1j}, y_{2j}, \ldots, y_{nj})$ be a vector that contains the same values as p_j, except that they are sorted. Let $\phi : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}$ be a function that, applied to an index in q_j, returns the position of the corresponding element in p_j. In other words, for all $l, h = 1, 2, \ldots, n : y_{lj} = x_{hj}$ if and only if $\phi(l) = h$. Then, the *partial Slope-Based Complexity* value on the j^{th} dimension is defined as:

$$pc_j = \begin{cases} \frac{\sum_{i=1}^{n-2} \left| \frac{g_{\phi(i+1)} - g_{\phi(i)}}{y_{(i+1)j} - y_{ij}} - \frac{g_{\phi(i+2)} - g_{\phi(i+1)}}{y_{(i+2)j} - y_{(i+1)j}} \right|}{n-2} & \text{if } n \geq 3 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Finally, the complexity measure is defined as the sum of the partial complexities on all the dimensions of the feature space, i.e.:

$$Complexity = \sum_{j=1}^{k} pc_j \tag{4.3}$$

It is worth noting that it is quite common for certain fitness cases to have the same value in some feature j. In such situations, when the sorted fitness cases and their corresponding output values are arranged based on that feature, multiple output values will be obtained for a single fitness case. To address this, the median output value is computed, and all fitness cases with that value are unified into a single observation. The pseudocode for the calculation of this complexity measure can be found in Algorithm 3.

This definition of SBC differs slightly from the original proposal in Vanneschi et al. (2010). In the original paper, partial complexity was calculated across all dimensions of the feature space. However, in this definition, partial complexity is not calculated for dimensions unused in the function. This change is aimed at leaving out dimensions that do not contribute to the function's complexity. Additionally, while the original method summed the absolute differences between slopes of consecutive segments to compute partial complexity, the approach used in this work takes the average of these absolute differences instead. This adjustment was made to ignore the effect of variations in the number of fitness cases, making the measure neutral to the size of the dataset. To calculate this average, the total number of observations is used before assessing whether there are multiple fitness cases with the same value in feature j.

This approach is chosen because the presence of multiple observations with the same value in feature j indicates that the feature is highly constant, contributing little to complexity. Using the number of unique fitness cases after the adjustment would fail to properly account for this behavior, leading to an overestimation of partial complexity.

Algorithm 3: Slope-Based Complexity

Data: Individual ind, Dataset $dataset$
Result: Complexity $complexity$
$N \leftarrow length(dataset)$;
$complexity \leftarrow 0$;
$used\_features \leftarrow$ features used by ind;
foreach $j \in used\_features$ **do**
  $p_j \leftarrow$ unique values of feature j in $dataset$;
  $outputs \leftarrow \{\}$;
  foreach $v \in p_j$ **do**
    $outputs_v \leftarrow$ outputs for observations with value v in feature j;
    if $|outputs_v| > 1$ **then**
      Add median of $outputs_v$ to $outputs$;
    else
      Add value in $outputs_v$ to $outputs$;
    end
  end
  $q_j \leftarrow$ indexes of values in p_j after being sorted in ascending order;
  $pc_j \leftarrow 0$;
  for $i \leftarrow 1$ **to** $|q_j| - 2$ **do**
    $s_i \leftarrow \frac{(outputs[q_j[i+1]] - outputs[q_j[i]])}{(p_j[q_j[i+1]] - p_j[q_j[i]])}$
    $s_{i+1} \leftarrow \frac{(outputs[q_j[i+2]] - outputs[q_j[i+1]])}{(p_j[q_j[i+2]] - p_j[q_j[i+1]])}$
    $pc_j \leftarrow pc_j + |s_i - s_{i+1}|$;
  end
  if $|p_j| > 2$ **then**
    $pc_j \leftarrow \frac{pc_j}{N-2}$
  else
    $pc_j \leftarrow 0$
  end
  $complexity \leftarrow complexity + pc_j$;
end
return $complexity$;

4.3 Complexity Minimization Strategies

Several methods have been proposed for penalizing individuals on a GP run based on specific criteria, with both error and complexity being our focus in this work. Regardless of the method used, they all share a common goal: influencing the selection of individuals in the population, the main mechanism for guiding the evolution. In this work, the focus is to select individuals with both low error and low complexity, ensuring that the selected individuals are not only accurate but also simple and less prone to overfitting.

One straightforward approach to penalize complex models involves directly modifying the fitness function to minimize complexity alongside error. This typically involves adding a penalty term multiplied by a penalty factor to the fitness function. This approach has been widely used in the literature to optimize two objectives. For instance, the well-known parametric parsimony pressure technique introduced in the original Koza's work (Koza, 1992) is a form of bloat control that penalizes models that become excessively large by imposing a size penalty in the fitness function. Although simple, this approach has the disadvantage of requiring the definition of the penalty factor, which can be challenging. The choice of this parameter defines the balance between the two objectives and is usually difficult to tune.

Alternatively, one can use a traditional multi-objective optimization algorithm. This approach is more complex, but has the advantage of not requiring the definition of a penalty factor. This type of algorithms are designed to find the best trade-off between more than one objective during evolution. However, selecting the final best individual from the Pareto-Front, which represents the optimal trade-off, is not straightforward.

Another commonly used method is the Double Tournament selection. This approach has been explored as a diversity control method (Brameier & Banzhaf, 2002), as well as a non-parametric parsimony pressure method for bloat control (Luke & Panait, 2002a). Nevertheless, it can be applied to optimize any two objectives. It works by running the traditional tournament selection twice, once for each objective. Despite being simple and not needing the definition of a penalty term, it still requires parameter tuning, such as the tournament sizes, second tournament probability and tournaments order. The need for parameter tuning might be seen as a disadvantage, but it also offers flexibility to explore the impact of different combinations.

A more complex method, introduced in this work as the Modified Multi-Objective Tournament Scheme, elegantly integrates both objectives into a single selection step. This method is an adaptation of the "Variance based selection scheme" proposed by Azad and Ryan (2011). It allows for the selection of an individual based solely on one of the objectives if it compensates enough for the decline on the other objective, without giving away too much on this other objective. Even though it has the advantage of being parameter-free, it may oversimplify the trade-offs between objectives, struggling to defining a good balance between them. Furthermore, this method relies

on the calculation of distances between individuals in the objective space, making it crucial to normalize the values of the objectives. This process becomes computationally expensive, as it requires evaluating all individuals in each generation to find the minimum and maximum values of each objective for min-max scaling. More details on this process will be provided later.

Table 4.1 summarizes the advantages and disadvantages of each of the presented complexity minimization methods. In this work, the focus will be directed towards implementing and evaluating the Double Tournament selection and the Modified Multi-Objective Tournament Scheme. This choice is based on the fact that these two methods show a balanced trade-off between benefits and drawbacks.

Table 4.1: Complexity minimization strategies comparison.

Method	Advantages	Disadvantages
Penalty Term	Easy to implement	Penalty factor tuning
Multi-Objective Optimization	No parameters to tune	Having to manually select an individual from the Pareto-Front
Double Tournament	Easy to implement Flexible	Several parameters to tune
Modified Multi-Objective Selection Scheme	No parameters to tune	Over-simplification of trade-offs Heavy to compute

4.3.1 Double Tournament

The Double Tournament (DT) algorithm (Luke & Panait, 2002a) is a simple yet effective two-step process for multi-objective optimization. By employing this approach, the algorithm can effectively evaluate candidates based on two objectives, ensuring a balanced optimization strategy.

Initially, several independent tournaments are run using the first objective as selection criteria, where competitors are randomly drawn from the population. The winners from these initial tournaments then compete in a final tournament, where the winning individual is selected based on the second objective. This algorithm has parameters for tournament sizes, a switch for the order of objectives, and a probability parameter determining the frequency of secondary objective selection.

For the purposes of this study, the two objectives being minimized are predictive error and complexity. Algorithm 4 details the double tournament selection algorithm implemented in this work. Since this algorithm relies on the original single tournament selection, the reader is referred to section 2.3.1.4 for a detailed explanation of

this algorithm.

It is important to note that, even if complexity is considered as the first objective, when the probability of selecting by the secondary objective is not achieved, the algorithm always prioritizes low error. It does this by returning the winner of a simple tournament selection based on fitness. This choice is justified by the fact that minimizing error is more important than minimizing complexity for the success of evolution. Choosing individuals based only on complexity could be too extreme and lead to bad overall performance.

Algorithm 4: Double Tournament Selection

Data: Population pop, Fitness tournament size S_f, Complexity tournament size S_c, Switch $switch$, Probability of selecting by second objective DT_prob
Result: Tournament winner $winner$
if $rand() < DT\_prob$ **then**
  $winners\_list \leftarrow \{\}$;
  if $switch$ **then**
    for $i \leftarrow 1$ **to** S_f **do**
      $selected\_ind \leftarrow$ TournamentSelection(pop, S_c, "complexity");
      $winners\_list \leftarrow winners\_list \cup selected\_ind$;
    end
    $winner \leftarrow$ Individual that minimizes fitness in $winners\_list$;
  else
    for $i \leftarrow 1$ **to** S_c **do**
      $selected\_ind \leftarrow$ TournamentSelection(pop, S_f, "fitness");
      $winners\_list \leftarrow winners\_list \cup selected\_ind$;
    end
    $winner \leftarrow$ Individual that minimizes complexity in $winners\_list$;
  end
else
  $winner \leftarrow$ TournamentSelection(pop, S_f, "fitness");
end
return $winner$;

4.3.2 Modified Multi-Objetive Tournament Scheme

The Modified Multi-Objective Tournament Scheme (MMOTS) implemented in this work is inspired by the technique described in Azad and Ryan (2011). This approach adopts a single-objective strategy despite optimizing two objectives. It is based on the idea that an individual may be chosen based on one objective alone if it sufficiently

compensates for decline in the other objective, while ensuring that this decline is not too significant.

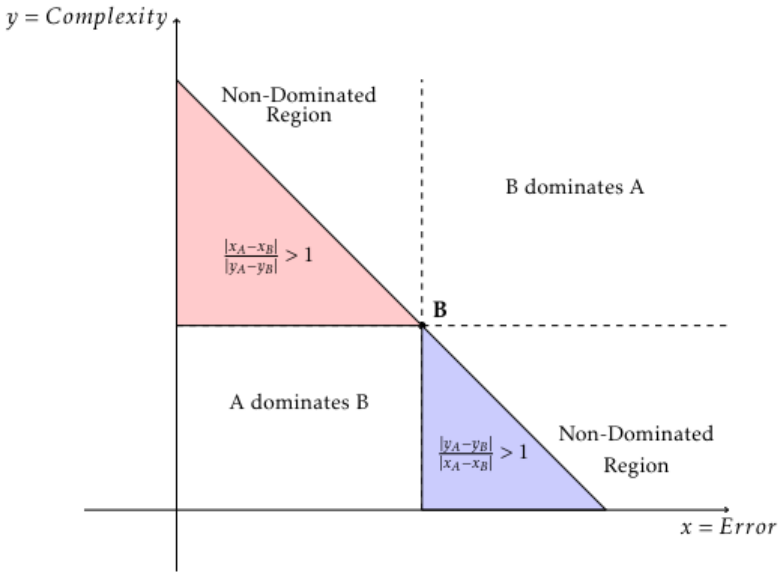
In this scheme, two randomly selected candidate models, A and B, are compared in a stepwise manner. If a decision cannot be reached in one step, the process moves to the next steps. It begins by determining if A dominates B (meaning A is at least equal to B in one objective and better in the other). If so, A is selected. If not, the algorithm checks if B dominates A, choosing B if so. When neither model dominates the other, the algorithm determines whether A's improvement in one objective compensates for the decline in the other objective, ensuring the trade-off is reasonable without sacrificing too much in that other objective.

To understand how this decision is made, consider the error-complexity space in Fig. 4.5(a). In the figure, point B represents an individual competing against A, characterized by its error and complexity values. If no decision was made in the previous step (i.e., A does not dominate B and B does not dominate A), individual A is located somewhere in the non-dominated regions.

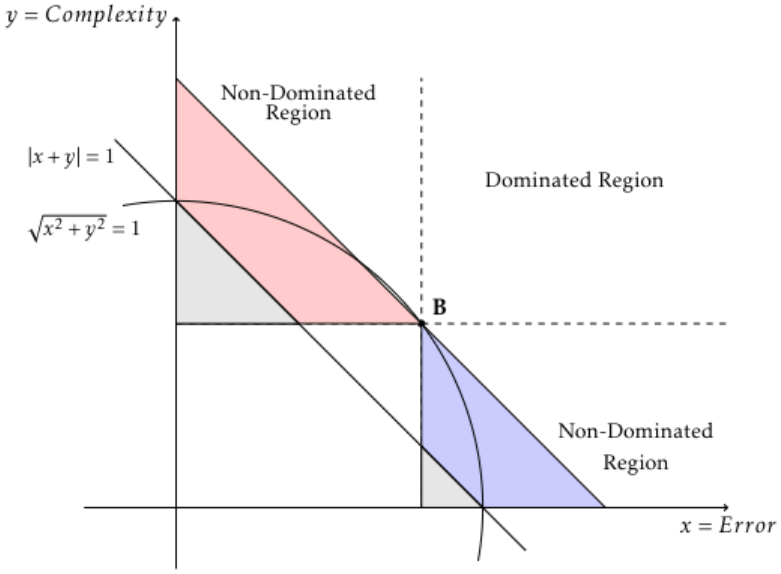
Starting with the upper non-dominated region: if A falls within this region, it means that A is better than B in terms of error, but worse in terms of complexity. In this case, the algorithm first determines whether A's increase in complexity is compensated by the decrease in error. This trade-off is evaluated by calculating the ratio of the change in error to the change in complexity between A and B: $\frac{\Delta error}{\Delta complexity} = \frac{|x_A - x_B|}{|y_A - y_B|}$. A ratio greater than 1 means that the improvement in error is more significant than the decline in complexity. The region where this ratio is greater than 1 is shaded in red in Fig. 4.5(a). If A falls within this region, its improvement in error outweighs the decline in complexity.

A similar reasoning applies to the lower non-dominated region. If A falls within this region, it is worse than B in terms of error but better in terms of complexity. To determine if A's improvement in complexity outweighs its decline in error, a similar ratio is used, this time with the change in complexity between A and B in the numerator and the change in error as the denominator: $\frac{\Delta complexity}{\Delta error} = \frac{|y_A - y_B|}{|x_A - x_B|}$. If this ratio is greater than 1, it means that the decrease in complexity is more significant than the increase in error. This region is shaded in blue in Fig. 4.5(a).

By calculating these ratios, it is possible to determine if the improvement of individual A in one objective compensates for the decline in the other. However, MMOTS takes it step further. Instead of immediately accepting A if it falls within the red or blue regions, an even more exclusive subregion within these regions is defined, ensuring a greater balance between the decline in one objective and the improvement in the other. This makes it possible to determine whether A improves over B in one objective without sacrificing too much in the other. To achieve this, the rectilinear distance of A ($|error_A + complexity_A|$) is compared to the euclidean distance of B ($\sqrt{error_B^2 + complexity_B^2}$). If the rectilinear distance of A is smaller than the euclidean



(a) Portions of the non-dominated regions where improvement in one objective outweighs the decline in the other.



(b) Portions of the non-dominated region where the rectilinear distance of A is smaller than the euclidean distance of B. These subregions of the red and blue regions are closer to the knee of the Pareto-Front.

Figure 4.5: The Modified Multi-Objective Tournament Scheme selection criteria in case of non-dominance.

distance of B, A is selected.

This scenario is illustrated in Fig. 4.5(b), where the same point B as in Fig. 4.5(a) is shown. B is assumed to have euclidean distance 1 for simplicity. The straight line represents points with a rectilinear distance of 1, with all the individuals below this line having a smaller rectilinear distance than B's euclidean distance.

The gray shaded areas contain points from the non-dominated regions with a rectilinear distance smaller than B's euclidean distance. It is evident that all individuals in the gray regions fall within the red and blue regions, indicating that their improvement in one objective outweighs the decline in the other. However, these gray regions are even more selective, containing only points where the balance between the two objectives is more reasonable. Accepting A only if it falls within these gray regions, rather than simply the broader red and blue regions, brings us closer to the knee of the Pareto Front (Branke et al., 2004), where the best trade-off between the two objectives is found. This avoids the selection of individuals that are much better in one objective but much worse in the other, which is not desirable.

Therefore, there is no need to explicitly calculate the ratios of the changes in error and complexity between A and B to see if A's improvement in one objective compensates for the decline in the other. Instead, the algorithm simply checks if the rectilinear distance of A is smaller than euclidean distance of B, since all points satisfying this inequality are already within the regions with a good trade-off between the two objectives (the red and blue regions). This approach is more efficient, avoids unnecessary calculations and even guarantees a better trade-off between the two objectives.

Finally, if A's rectilinear distance is not smaller than B's euclidean distance, the evaluation repeats with A and B's roles reversed: if B's rectilinear distance is smaller than A's euclidean distance, B is selected. If still undecided, the model with the smaller complexity is chosen.

In the original paper, the authors use this selection scheme to minimize variance alongside error, as they believe the variance of a function's outputs is a good approximation for its complexity. However, in this work, the Slope-Based Complexity (SBC) measure is used as the second objective instead. Additionally, the original algorithm was modified to account for the differences in range between the two objectives. This step is extremely important because the algorithm relies on the calculation of distances between individuals in the objective space. If the objectives are not normalized and their domains vary significantly, the algorithm may become biased towards the objective with the larger range. To address this, the objectives are normalized using min-max scaling, which involves finding the minimum and maximum values of each objective in the population and scaling the values of each individual accordingly, bringing them into the [0,1] range. In this work, normalization is repeated for each generation's population, as complexity and error values change significantly over generations.

The pseudocode for this Modified Multi-Objective Tournament Scheme can be

found in Algorithm 5.

Algorithm 5: Modified Multi-Objective Tournament Scheme

Data: Population pop
Result: Tournament winner $winner$
$indA \leftarrow$ Random individual from pop ;
$indB \leftarrow$ Random individual from pop;
fA $\leftarrow MinMaxScaling(fitness(A))$;
$fB \leftarrow MinMaxScaling(fitness(B))$;
$cA \leftarrow MinMaxScaling(SBC(A))$;
$cB \leftarrow MinMaxScaling(SBC(A))$;
switch 1 **do**
  case $(fA \leq fB \wedge cA < cB) \vee (fA < fB \wedge cA \leq cB)$ **do**
    $winner \leftarrow indA$;
  end
  case $(fB \leq fA \wedge cB < cA) \vee (fB < fA \wedge cB \leq cA)$ **do**
    $winner \leftarrow indB$;
  end
  case $|fA + cA| < \sqrt{fB^2 + cB^2}$ **do**
    $winner \leftarrow indA$;
  end
  case $|fB + cB| < \sqrt{fA^2 + cA^2}$ **do**
    $winner \leftarrow indB$;
  end
  otherwise do
    $winner \leftarrow$ Individual that minimizes complexity;
  end
end
return $winner$;

5

Experimental Study

This chapter details the experimental study of the proposed methodology. Section 5.1 describes the case studies used in the experiments. Section 5.2 covers the experimental settings, including parameter configurations and evaluation metrics. Finally, Section 5.3 presents and discusses the experimental results, quantifying the performance of the proposed methods in terms of generalization and model quality, the impact of different hyperparameters on the *DT* algorithm's performance, and an in-depth analysis of the evolved models and the relationship between complexity and overfitting.

5.1 Datasets

The datasets used in the experiments are summarized in Table 5.1. These datasets were selected to cover a wide range of domains and characteristics, including different numbers of instances and features. All have been extensively used as case studies for Symbolic Regression applications, a ML approach in which both the parameters and structure of a model are optimized.

Table 5.1: Datasets used in the experiments.

Dataset	Number of Features	Number of Observations
USCrime	13	47
Pollution	15	60
Concrete	8	1030
Istanbul	7	536
BHousing	13	506
PPB	626	131
LD50	626	234
Bioavailability	241	359

United States Crime (USCrime)

The United States Crime dataset is a result of a study conducted by the FBI's Uniform Crime Report and other government agencies to investigate the relationship between the crime rate and various demographic variables. It contains crime-related and demographic statistics for the 47 US states in the year of 1960. This dataset is part of the Penn Machine Learning Benchmarks (PMLB) (Olson et al., 2017), a framework for benchmarking ML and SR algorithms containing well-known, real-world datasets from UCI and OpenML (Cava et al., 2021).

The target in this dataset is a continuous variable representing the proportion of families, out of every 1000, whose income falls below half of the median income. This value reflects the socio-economic status of families within each state. All variables are described in 5.2, along with relevant statistics.

Table 5.2: USCrime variables and statistics.

Variable	Description	Max	Min	Avg	Stdev
R	Reported offenses per million population	199.3	43.9	37.2	91.92
Age	Males of age 14-24 per 1000 population	177.0	119.0	13.03	0.34
S	Indicator variable for Southern states	1.0	0.0	0.48	0.34
Ed	Mean number of years of schooling x 10 for persons of age 25 or older	122.0	87.0	11.29	106.43
Ex0	1960 Police expenditure per capita	166.0	46.0	29.25	84.74
Ex1	1959 Police expenditure per capita	157.0	41.0	27.99	79.81
LF	Labor force participation rate per 1000 civilian urban males age 14-24	641.0	480.0	41.6	561.68
LF	Labor force participation rate per 1000 civilian urban males age 1071.0	1071.0	934.0	29.51	984.89
M	Males per 1000 females	168.0	3.0	39.39	37.57
N	State population size in hundred thousands	168.0	3.0	39.39	37.57
NW	Non-whites per 1000 population	423.0	2.0	92.99	90.11
U1	Unemployment rate of urban males per 1000 of age 14-24	142.0	70.0	17.83	95.64
U2	Unemployment rate of urban males per 1000 of age 35-39	58.0	20.0	8.63	34.15
W	Median value of transferable goods and assets or family income in tens of dollars	689.0	288.0	91.71	528.72
Target	The number of families per 1000 earning below 1/2 the median income	276.0	126.0	39.71	192.81

Air pollution and mortality rate (Pollution)

The Polution dataset, introduced by McDonald and Schwing, 1973, explores the correlation between air pollution levels and mortality rates. It includes a range of environmental and demographic factors potentially linked to mortality. The target variable measures the total age-adjusted mortality rate per 100,000 individuals. The variables collectively represent the complex interplay between environmental and demographic factors and their impact on mortality rates. This dataset is also part of the Penn Machine Learning Benchmarks (Olson et al., 2017).

Table 5.3: Pollution variables and statistics.

Variable	Description	Max	Min	Avg	Stdev
PREC	Average annual precipitation (in)	60.0	10.0	10.88	36.65
JANT	Jan average temperature (°F)	67.0	12.0	10.58	34.72
JULT	July average temperature (°F)	85.0	63.0	4.88	74.53
OVR65	% of 1960 SMSA population 65+	11.8	5.60	1.58	8.82
POPN	Average household size	3.49	2.92	0.13	3.26
EDUC	Median school years completed by those over 22	12.3	9.0	0.93	11.02
HOUS	Percentage of housing units which are sound and with all facilities	9.070	69.20	5.19	81.22
DENS	Urban population density	9699.0	1441.0	1666.79	3922.85
NONW	Urban non-white population %	36.70	0.80	8.68	10.99
WC	Percentage of employed in white collar occupations	59.70	37.5	4.71	47.03
POOR	Percentage of families with income < $3000	26.40	9.40	3.94	14.13
HC	Relative hydrocarbon pollution potential	648.0	1.0	93.28	39.77
NOX	Relative NOx pollution potential	319.0	1.0	46.12	21.85
SO2	Relative SO2 pollution potential	206.0	1.0	48.19	45.33
HUMID	Annual average % relative humidity at 1pm	73.0	38.0	5.71	57.75
Target	Total age-adjusted mortality rate per 100,000	1113.16	790.73	66.06	931.65

Concrete Strength (Concrete)

The Concrete dataset is a popular dataset for regression analysis and predictive modeling in the field of civil engineering, and was introduced by Yeh (1998) to predict the strength of high-performance concrete using artificial neural networks. It was obtained from the UCI Machine Learning Repository (Kelly et al., 2023), a popular resource for getting open source and free datasets for ML applications.

Cement is a key component in concrete, a widely used construction material. The choice of concrete mix depends on factors such as strength, appearance, and local regulations, with high-performance concrete (HPC) representing a more complex version of concrete. Accurately predicting the compressive strength of concrete is crucial for ensuring the structural integrity of buildings and infrastructure, which is influenced by factors like the type of cement used, the water-to-cement ratio, aggregate properties, and curing conditions. Reliable techniques to predict concrete strength are essential for optimizing concrete mixtures, ensuring structural safety, and reducing construction costs.

Previous studies using different Machine Learning techniques, such as Artificial Neural Networks (Lee, 2003), Genetic Algorithms (Akkurt et al., 2003), Fuzzy Logic (Demir, 2005), and Genetic Programming (Castelli et al., 2013), have been conducted using this dataset. All its variables describe either an amount of a component in the concrete mixture or another mixture characteristic. Statistics for the variables are shown in Table 5.4.

Table 5.4: Concrete variables and statistics.

Variable	Max	Min	Avg	Stdev
Cement (kg/m^3)	540.0	102.0	104.51	281.17
Fly Ash (kg/m^3)	359.4	0.0	86.28	73.9
Blast Furnace Slag (kg/m^3)	200.1	0.0	64.0	54.19
Water (kg/m^3)	247.0	121.8	21.35	181.57
Superplasticizer (kg/m^3)	32.2	0.0	5.97	6.2
Coarse aggregate (kg/m^3)	1145.0	801.0	77.75	972.92
Fine aggregate (kg/m^3)	992.6	594.0	80.18	773.58
Age of testing	365.0	1.0	63.17	45.66
Target	82.6	2.33	16.71	35.82

Istanbul Stock Exchange (Istanbul)

Similar to the Concrete dataset, the Istanbul dataset, introduced by Akbilgic et al., 2013 to determine the optimal lags of international indicators predicting the direction of movement of the Istanbul Stock Exchange index, is also part of the UCI Machine Learning Repository (Kelly et al., 2023).

It contains multivariate time-series data on the returns of the Istanbul Stock Exchange between 2009 and 2011, which include daily prices of other stock market indices that influence the Istanbul Stock Exchange index (ISE 100 index). With the target variable being the ISE 100 index returns in US Dollars, predicting this variable is crucial for investors to make informed decisions, optimize portfolios, manage risk effectively, gain insights into the economy, and forecast market trends with greater precision. The dataset includes 536 instances and 8 features, which are described in Table 5.5.

Table 5.5: Istanbul variables and statistics.

Variable	Description	Max	Min	Avg	Stdev
SP	SP500 return index	0.07	-0.05	0.01	0.0
DAX	Stock market return index of Germany	0.06	-0.05	0.01	0.0
FTSE	Stock market return index of UK	0.05	-0.05	0.01	0.0
NIK	Stock market return index of Japan	0.06	-0.05	0.01	0.0
BVSP	Stock market return index of Brazil	0.06	-0.05	0.02	0.0
EU	MSCI European index	0.07	-0.05	0.01	0.0
EM	MSCI emerging markets index	0.05	-0.04	0.01	0.0
Target	Istanbul stock exchange national 100 index	0.1	-0.08	0.02	0.0

Boston Housing (BHousing)

The Boston Housing dataset is a compilation of data gathered by the U.S. Census Service, focusing on housing attributes in the Boston, Massachusetts area. This dataset, accessible through the StatLib archive (Kooperberg, 1997), has been widely used in research to evaluate and benchmark various algorithms.

Originally published by Harrison and Rubinfeld, 1978 as part of a research investigating methodological challenges in measuring willingness to pay for clean air using housing market data, this dataset is as a valuable resource for studying housing trends, property values, and related economic and environmental factors in the Boston region. Each instance in the dataset represents a neighborhood in Boston and the target variable is the median value of owner-occupied homes in $1000's.

Similarly to the other datasets, the Boston dataset been extensively used as a regression dataset in multiple research studies, for instance, works involving Support Vector Machines (Stitson et al., 1999), Ensemble Methods (Bhardwaj et al., 2020), and Evolutionary Computation (Zhou et al., 2001), (N. Lourenço et al., 2017). Details of the variables are shown in Table 5.6.

Table 5.6: BHousing variables and statistics.

Variable	Description	Max	Min	Avg	Stdev
CRIM	Per capita crime rate by town	88.98	0.01	8.6	3.61
ZN	Proportion of residential land zoned for lots over 25,000 sq.ft.y	100.0	0.0	23.32	11.36
INDUS	Proportion of non-retail business acres per town	27.74	0.46	6.86	11.14
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)	1.0	0.0	0.25	0.07
NOX	NOx concentration (ppm)	0.87	0.38	0.12	0.55
RM	Average number of rooms per dwelling	8.78	3.56	0.7	6.28
AGE	Proportion of owner-occupied units built prior to 1940	100.0	2.9	28.15	68.57
DIS	Weighted distances to five Boston employment centres	12.13	1.13	2.11	3.8
RAD	Index of accessibility to radial highways	24.0	1.0	8.71	9.55
TAX	Full-value property-tax rate per $10,000	711	187	168.54	408.24
PTRATIO	Pupil-teacher ratio by town	22.0	12.6	2.16	18.46
B	$1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town	396.9	0.32	91.29	356.67
LSTAT	% lower status of the population	37.97	1.73	7.14	12.65
Target	Median value of owner-occupied homes in $1000's	50.0	5.0	9.2	22.53

Plasma Protein Binding Levels (PPB)

The PPB dataset is a collection of data on the plasma protein binding levels of various drugs and toxic substances. It is a real-world dataset from the field of pharmacokinetics, which studies the absorption, distribution, metabolism, and excretion of drugs in the body.

Plasma protein binding (PPB) refers to the interaction between drugs and proteins in the blood, impacting the drug's distribution and activity in the body. Predicting this

binding level is crucial for understanding how much of the drug will be available to produce its therapeutic effects. Accurate prediction helps in optimizing drug dosing, efficacy, and safety profiles, making it a key aspect of drug development and clinical use (Ghafourian & Amin, 2013), (Li et al., 2021).

The traditional approach to predicting plasma protein binding levels, as well as other pharmacokinetics properties of drugs, involves using Quantitative Structure-Activity Relationship (QSAR) models, which establish a quantitative relationship between the structure of a molecule and its biological activity. However, in recent years, machine learning techniques have been increasingly used to predict pharmacokinetics properties, including plasma protein binding levels, due to their ability to handle complex relationships between molecular structures and biological activities.

The data was obtained by collecting information about a set of molecular structures and the corresponding PPB values using the same data as in Yoshida and Topliss (2000) and a public database of the food and drug administration (FDA) approved drugs and drug-like compounds (Wishart et al., 2006). Chemical structures of drugs were represented as Simplified Molecular Input Line Entry Specification (SMILES) codes and analyzed by software to extract 626 bi-dimensional molecular descriptors.

This dataset contains molecular descriptors of 131 different drugs, with the target variable being the plasma protein binding level. It has been subject of study in the field of Genetic Programming (Archetti, Lanzeni, Messina, & Vanneschi, 2007), (Archetti, Lanzeni, Messina, & Vanneschi, 2007), and other deep learning techniques (Li et al., 2021). Given the high number of features and abstract nature of the variables, no statistics or descriptions are provided for this dataset.

Median Oral Lethal Dose (LD50)

The LD50 is another dataset from the field of pharmacokinetics, focusing on the median oral lethal dose of various drugs and toxic substances, measured in rats. LD50 is a measure of the toxicity of a substance, representing the dose required to kill 50% of the test organisms. It is a critical parameter in drug development, safety assessment, and regulatory approval processes.

This dataset has been obtained from the same sources as the PPB dataset, and contains information on 234 different drugs, with the target variable being the median oral lethal dose. This data has also been explored in studies similar to the ones involving the PPB dataset, such Archetti, Lanzeni, Messina, and Vanneschi (2007) and Archetti, Lanzeni, Messina, and Vanneschi (2007), where Genetic Programming was used as the main approach.

Once again, due to the high number of features and abstract nature of the variables, no statistics or descriptions are provided for this dataset.

Human Oral Bioavailability (Bioavailability)

Human oral bioavailability (F%) is a critical measure in pharmacokinetics, representing the fraction of an orally administered drug that enters systemic circulation. Predicting this value is crucial in drug development, as it directly impacts dosing, efficacy, and safety. Higher oral bioavailability reduces the required drug dose and lowers the risk of side effects, while poor bioavailability can lead to reduced efficacy and unpredictable responses, affecting a drug's success in clinical trials.

Once again, this dataset was obtained from the same sources as the PPB and LD50 datasets, but using a different software to obtain the molecular descriptors. These include 241 bi-dimensional molecular descriptors for 359 different drugs, with the target variable being the human oral bioavailability.

This dataset was first introduced in a study related to GP (Archetti et al., 2006) and later used in research on bloat-free genetic programming (Silva & Vanneschi, 2012), before being included in the same studies as the PPB and LD50 datasets that were previously mentioned. As with the other datasets, no statistics or descriptions are provided for this dataset due to the high number and abstract nature of features.

5.2 Experimental Settings

The algorithms used in this study to simultaneously minimize complexity alongside error were evaluated for their ability to mitigate overfitting in GP. These strategies, DT and MMOTS, modify the selection step of GP evolution to consider two objectives rather than just one. In contrast, Standard GP (StdGP) solely focuses on minimizing error during evolution. The implementation of StdGP in this study uses a simple tournament selection mechanism (see Section 2.3.1.4 for implementation details), a common choice in GP due to its simplicity and effectiveness.

Variants of GP that use DT and MMOTS as the selection mechanisms were compared against each other and to the baseline results of StdGP, which does not employ complexity minimization strategies. Since DT and MMOTS only introduce changes to the selection step of the GP evolution, some common preprocessing steps and parameters were applied across all GP variants. Details of these shared preprocessing steps and parameter settings are described in Section 5.2.1, with specific modifications introduced by DT and MMOTS detailed in Sections 5.2.2 and 5.2.3, respectively.

5.2.1 Base

5.2.1.1 Preprocessing steps

The original datasets were partitioned into 30 random splits of size 70-30, with 70% of the data used for training and 30% for testing. The execution of this repeated hold-out method ensures the robustness of the results.

Following that, all partitions were preprocessed using Min-Max Scaling (Han et al., 2011), which standardizes the data to a fixed range of values from 0 to 1. Although scaling does not influence the performance of the GP evolution, its implementation is crucial for accurately computing the complexity measure used in this work. This process guarantees that all features are in the same scale, preventing potential biases towards features with higher values. The reader is referred to section 4.2 for more details on the SBC calculation.

5.2.1.2 Parameters Settings

Given the objective of this study is to compare different GP variants regarding their impact on the generalization ability of evolved models, some fixed hyperparameters were set for all variants. These were not optimized as that optimization is outside the scope of this work, but their values were chosen based on previous studies and best practices in GP. The parameters shared by all the GP variants are detailed in Table 5.7.

Table 5.7: Common hyperparameters to all GP variants.

Parameter	Value
Population size	100
Stopping condition	500 generations
Initialization	Ramped Half-and-Half (Koza, 1992)
Max initial tree depth	7
Max tree depth during evolution	17
Crossover	Koza's subtree crossover (Koza, 1992)
Mutation	Koza's subtree mutation (Koza, 1992)
Crossover rate	0.9
Mutation rate	0.1
Elitism	Used, with elite size 1
Primitive functions	+, -, *, //, where // returns 1 if division by 0
Terminals	One variable for each feature
Fitness	RMSE

5.2.2 Double Tournament

The Double Tournament is a selection mechanism aimed at reducing overfitting by selecting individuals based on their performance in two different objectives. One of the objectives is the fitness value, which measures the model's quality on the training

data. The other is the complexity measure, which approximates the curvature of the function, and is used to penalize overly complex models that tend to overfit.

Therefore, this approach introduces some additional hyperparameters. Table 5.8 details the specific values selected for these hyperparameters in the base GP variant using the Double Tournament as selection mechanism (from now on referred to as *DT*). It is important to note that these values were initially chosen based on intuition, and a more in-depth study was later conducted to assess their impact on the overall performance of the approach.

The double tournament probability was initially determined to be 1, in order to test the most extreme version of the approach, where all individuals are selected based on the two objectives scheme. The tournament order was set to fitness first and complexity second, as the main goal is to prioritize the selection of individuals with good predictive ability, and only penalize overly complex models afterwards. Finally, the tournament sizes were based on the total population size, with 4 individuals competing in terms of fitness, and the 2 winners of those tournaments competing in terms of complexity. Once again, this choice aligns with the main goal of selecting individuals based on their predictive ability first, and then choose the less complex one from those. A greater selection pressure in terms of fitness is ensured by using a larger tournament size for the first tournament, while a lower selection pressure in terms of complexity is obtained by using a smaller tournament size for the second tournament. This aims to avoid overly penalizing models and obtain a good predictive ability.

Table 5.8: Double Tournament base hyperparameters.

Parameter	Value
Double Tournament Probability	1
Tournament order	Fitness → complexity
First tournament size	4
Second tournament size	2

5.2.2.1 Study on Double Tournament Hyperparameters

Given the variety of hyperparameters of the Double Tournament algorithm, it is interesting to study their impact on the approach's performance. A series of experiments were conducted to assess the impact of the double tournament order, probability, and tournament sizes on the performance and generalization ability of the evolved models. When one of these hyperparameters is changed, all others are kept fixed at the values shown in Table 5.8, in order to isolate the impact of each specific parameter.

Tournament Order The order in which the tournaments are conducted can have a significant impact on the overall performance. For instance, the base version of the

Double Tournament (labeled *DT*) uses fitness as the first objective and complexity as the second. This order was chosen based on the intuition that the priority should be to select individuals with good predictive error first, and from those, chose the less complex one. Intuitively, it is possible that selecting the less complex individuals first could lead to a bias towards overly simple models, which may result in bad performing models. However, it is not possible to derive this conclusion solely based on intuition. Therefore, these two tournament orders are worth studying, while maintaining all other hyperparameters fixed. The version of the Double Tournament with the reversed tournament order will be referred to as *InvDT* moving forward.

Double Tournament Probability The double tournament probability is another hyperparameter that can have a significant impact on performance. The probability of 1, used in *DT*, means that all individuals are selected using the two objectives. However, this may be too extreme, leading to a bias towards overly simple models, which can affect evolution. Therefore, it is worth studying different probabilities to assess their impact on performance. The probabilities of 0.5 and 0.7 were chosen to explore different trade-offs between model quality and overfitting. These versions of *DT* will be referred to as *DT0.5* and *DT0.7*, respectively, while the corresponding versions of *InvDT* are labeled *InvDT0.5* and *InvDT0.7*.

Tournament Sizes *DT* uses a first tournament size of 4 and second of 2. These values were chosen based on the intuition that the first tournament should be larger to ensure a higher selection pressure based on fitness, while the second tournament should be smaller, to avoid overly penalizing models. However, these values were not optimized, and it is worth studying different tournament sizes to assess their impact on the evolution. The set of tested tournament sizes can be found in Table 5.9. Each of these GP variants will be referred to as *DT_X_Y*, where *X* and *Y* represent the tournament sizes for the first and second tournaments, respectively.

Table 5.9: Double Tournament tested sizes.

First Tournament Size	Second Tournament Size
2	2
2	4
4	2
4	4
10	2
10	4

5.2.3 Modified Multi-Objective Tournament Scheme

The Modified Multi-Objective Tournament Scheme introduced in Section 4.3.2 is another modification of the selection step in the GP evolution, aimed at reducing overfitting by minimizing error and complexity simultaneously. In contrast to the Double Tournament algorithm, this approach involves a single selection step, where individuals are chosen based on their performance across both objectives simultaneously.

As a result, this method introduces only one hyperparameter, the tournament size. The tournament size of the original proposal of the method is 2, which is justified by the way the method works. First, the approach tries to establish Pareto dominance between two individuals. If this criterion is not met, the selection is then based on a principle where gains in one objective should offset losses in the other, while still trying to maintain a good balance between the two objectives. If that is not possible, the individual with lower complexity is chosen.

The GP variant that uses this scheme as the selection mechanism will be referred to as *MMOTS* moving forward.

5.3 Experimental Results and Discussion

5.3.1 Proposed Approaches Comparison

Fig. 5.1 shows the train and test RMSE values for *DT* and *MMOTS*, compared to StdGP. These GP variants are described in sections 5.2.2 and 5.2.3, respectively. Each line represents the median RMSE of the best individual across the generations, based on the results of 30 independent runs.

Starting with *DT*, it is possible to observe that the performance of the best individual in the train set is similar to the one of StdGP in most use cases, although values tend to be slightly higher. However, by looking at the test set performance, one can see a significant improvement in the generalization ability. The test RMSE values are consistently lower than the ones of StdGP, and always closely follow the train set performance. In fact, one can even observe that the test set performance is better than the train set in the LD50 dataset, which indicates the absence of overfitting. Another interesting example is the Concrete dataset. A slight decrease in train performance can be observed compared to the StdGP. However, it is important to note that StdGP generalizes particularly well in this dataset. This is important to point out because, even though an overfitting avoidance method is employed in a dataset in which StdGP does not overfit the training data, the generalization ability is not affected and the decrease in performance is not very significant. These results indicate that *DT* is effective in reducing overfitting in all datasets where StdGP overfits, and does not negatively impact generalization in datasets where StdGP already generalizes well.

Similarly, *MMOTS* also has a significant impact on overfitting reduction. As it is possible to see in the figure, the train and test performance throughout generations

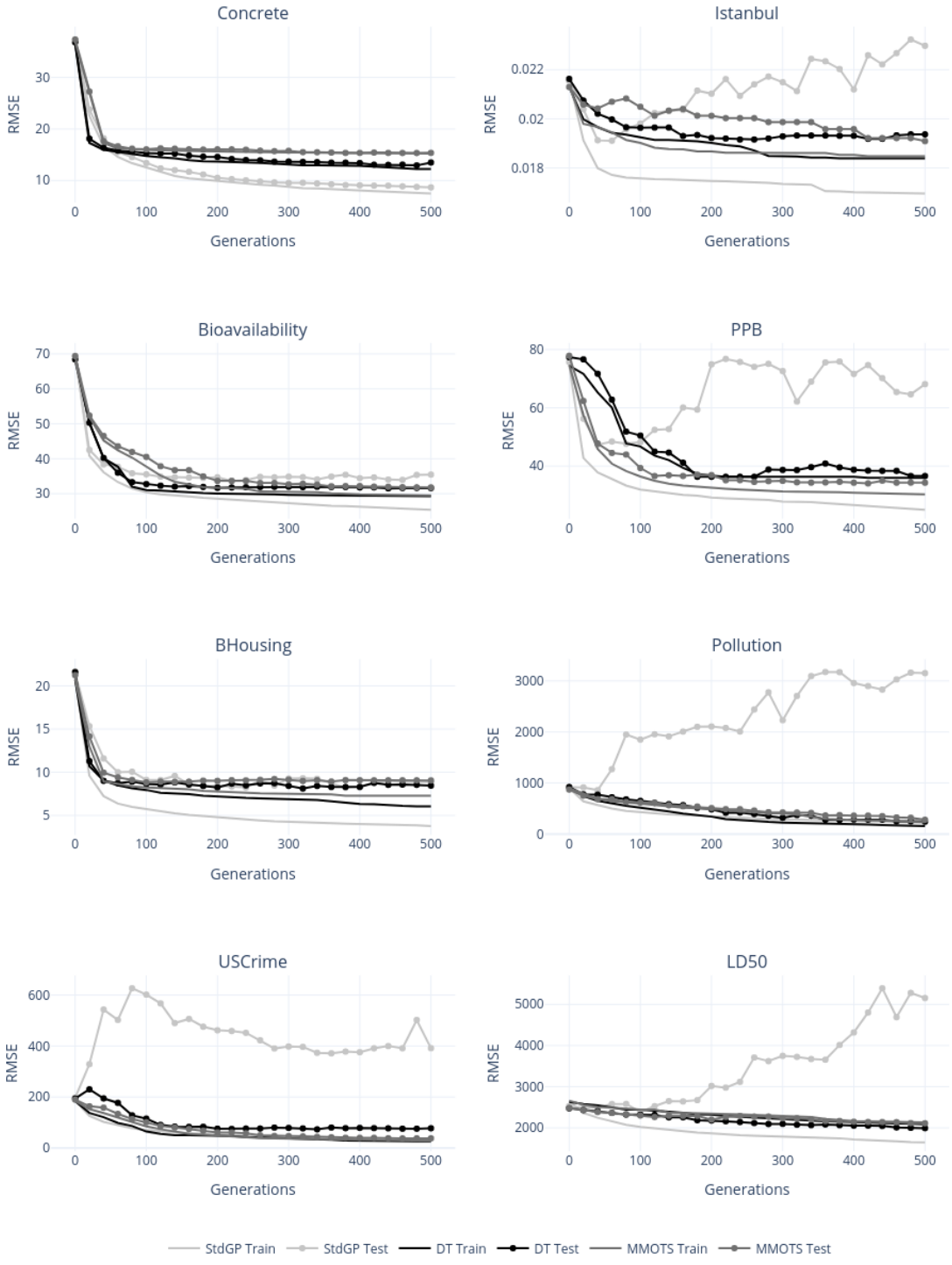


Figure 5.1: Median best-of-run performance over generations using StdGP (light gray), *DT* (black) and *MMOTS* (dark gray).

remain close, indicating a reduction in overfitting compared to StdGP. The performance of *MMOTS* and *DT* are very similar. Just like *DT*, the *MMOTS* variant also leads to a slight decrease in train and test performance on the Concrete dataset, but the generalization ability is not affected. Furthermore, the near absence of overfitting is evident in the USCrime, Pollution and LD50 datasets, as the train and test RMSE lines almost overlap. These results indicate that *MMOTS* is also effective in reducing overfitting in datasets where StdGP overfits, without impacting generalization where StdGP performs well.

All GP variants were compared in terms of statistical significance using Wilcoxon signed-rank test, a non-parametric statistical hypothesis test used to compare two related samples or paired data. This test gives robust results, even when the data does not meet the assumptions of normality required for parametric tests, like the t-test. The measure used in this statistical test is the difference between test and train RMSE in the last generation of the GP evolution, as it is a commonly used approximation for overfitting. Results are shown in Table 5.10, and highlighted values show statistical significance at a 0.05 level.

Table 5.10: P-values of the Wilcoxon test for the difference between test and train error on the last generation in StdGP, *DT* and *MMOTS*. Highlighted values show statistical significance.

	USCrime	Pollution	Concrete	Istanbul	BHousing	PPB	LD50	Bioavailability
StdGP vs *DT*	**0.000**	**0.000**	**0.022**	**0.000**	0.096	**0.000**	**0.000**	**0.000**
StdGP vs *MMOTS*	**0.000**	**0.000**	**0.001**	**0.000**	0.067	**0.000**	**0.000**	**0.000**
DT vs *MMOTS*	**0.028**	0.205	0.839	**0.002**	0.245	0.952	0.050	0.114

The statistical test shows that, considering only the last generation of the GP evolution, both *DT* and *MMOTS* are effective in reducing overfitting. The p-values indicate that the difference in generalization ability between the proposed approaches and StdGP is statistically significant in the majority of the datasets. On the contrary, *DT* and *MMOTS* are mostly not significantly different from each other, which indicates that, regarding overfitting reduction, both approaches are effective in a similar way.

In addition to the statistical test that focuses on the difference in test and train error at the end of GP evolution, Fig. 5.2 illustrates how the gap between test and train RMSE changes throughout the generations. This graph helps in better understanding how well the evolved models generalize and how consistent the results are throughout the evolution. Despite some fluctuations seen in these differences over generations,

both *DT* and *MMOTS* effectively reduce overfitting, maintaining a consistently lower gap between test and train RMSE compared to the StdGP version.

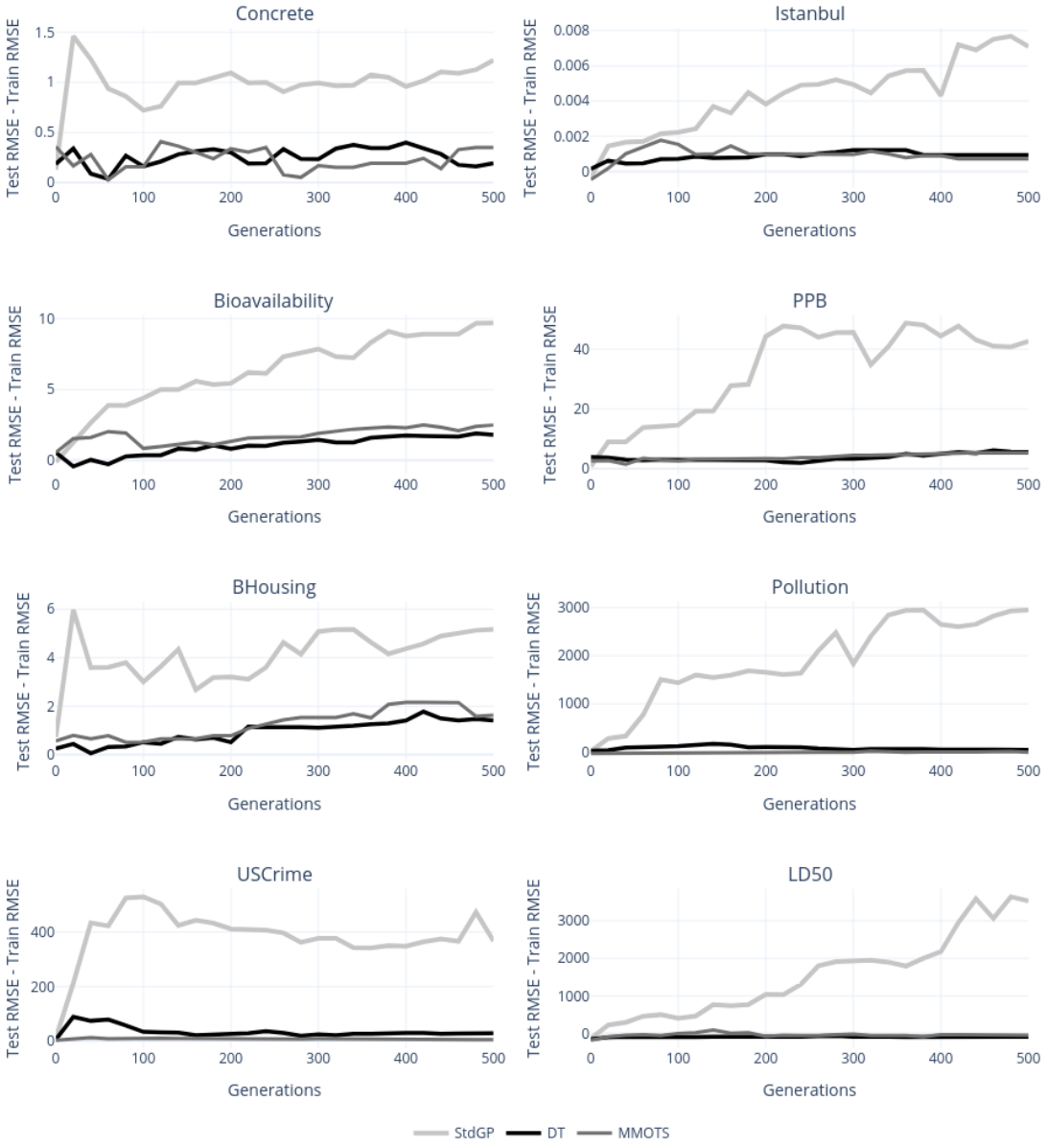


Figure 5.2: Difference between test and train error (proxy for overfitting) throughout generations for StdGP, *DT* and *MMOTS*.

In conclusion, both *DT* and *MMOTS* are effective in reducing overfitting in GP, while maintaining good generalization ability. The results show that the proposed methods are robust and consistently improve the generalization ability of the evolved models, without negatively impacting their predictive performance.

5.3.2 Impact of Double Tournament Hyperparameters

5.3.2.1 Tournaments Order

The order in which the tournaments are conducted has a significant impact on the performance of the Double Tournament approach.

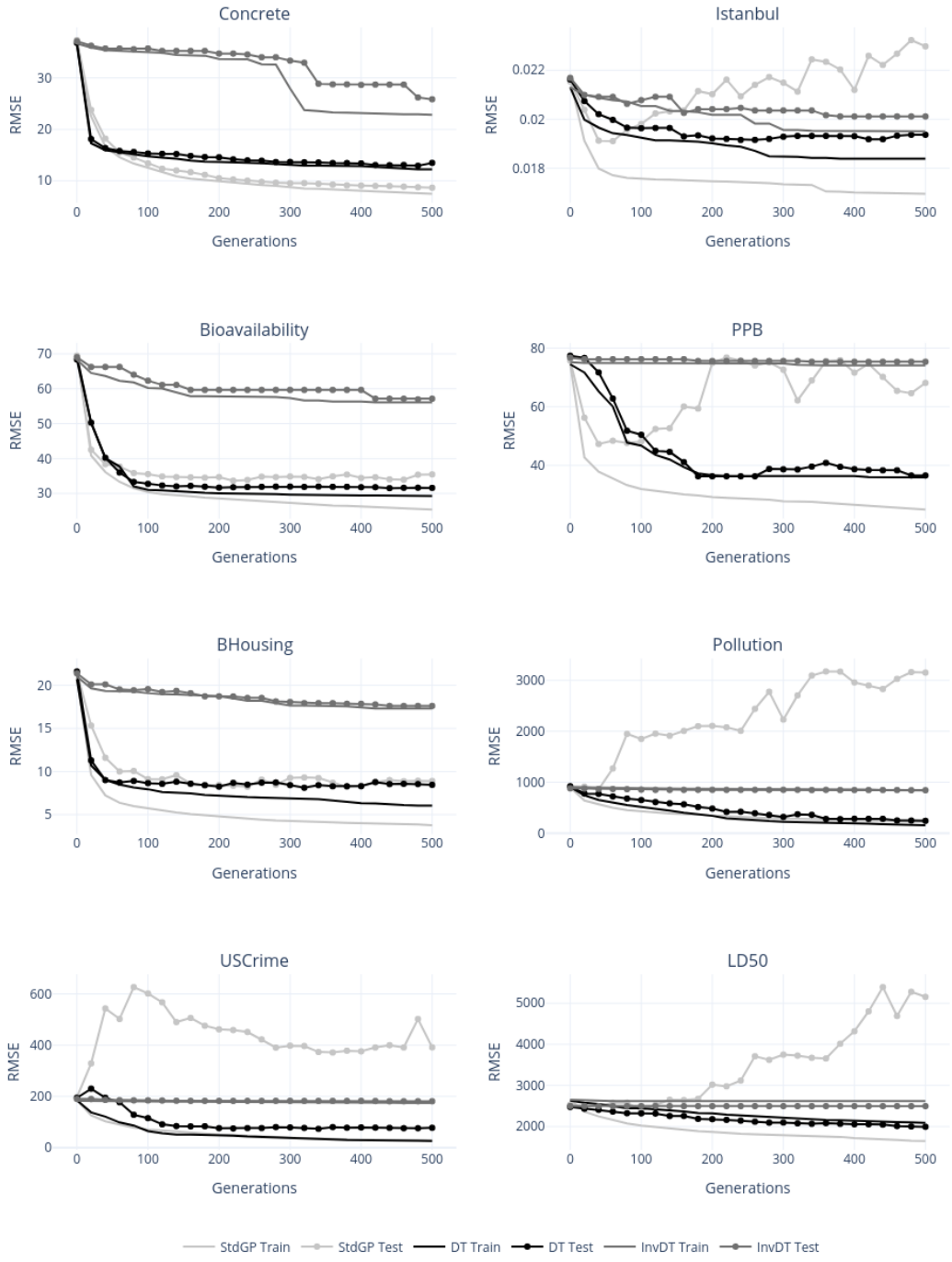


Figure 5.3: Median best-of-run performance throughout generations using StdGP (light gray), *DT* (black) and *InvDT* (dark gray).

Fig. 5.3 shows that the GP variant that uses the Double Tournament selection with complexity as the first objective and fitness as the second, called *InvDT*, also yields good results in terms of overfitting reduction comparing to StdGP. This is evident by the closeness of the train and test RMSE lines throughout the generations. However, the performance of the best-of-run individual in both train and test sets is consistently worse when compared to *DT*, which uses fitness as the first objective and complexity as the second.

InvDT shows signs of underfitting in all datasets, as evolution converges early or too slowly, resulting in suboptimal models that are not able to capture the patterns in data and have poor predictive performance. These results align with the initial intuition that prioritizing simpler individuals first, and then choosing the best performing ones can introduce a bias towards excessively simple models, resulting in bad performance.

Nevertheless, it is important to remember that this approach was tested using a double tournament probability of 1, which is its most extreme version. Therefore, it is possible that the results would be different if a less extreme version was employed. This hypothesis is explored in the next section.

Table 5.11: P-values of the Wilcoxon test for the difference between test and train error on the last generation in StdGP, *DT* and *InvDT*.

	USCrime	Pollution	Concrete	Istanbul	BHousing	PPB	LD50	Bioavailability
StdGP vs *DT*	**0.000**	**0.000**	**0.022**	**0.000**	0.096	**0.000**	**0.000**	**0.000**
StdGP vs *InvDT*	**0.000**	**0.000**	0.967	**0.000**	**0.000**	**0.002**	**0.000**	**0.000**
DT vs *InvDT*	**0.005**	0.006	0.067	0.191	**0.000**	**0.008**	0.164	0.903

InvDT was also compared to *DT* and StdGP regarding statistical significance, using the Wilcoxon signed-rank test with a statistical significance level of 0.05. The results, shown in Table 5.11, indicate that, in the majority of datasets, both versions of the double tournament are effective in maintaining the generalization ability in the last generation of the GP evolution comparing to StdGP. Conversely, *DT* and *InvDT* are not significantly different from each other in most cases, which indicates that, regarding overfitting reduction, the order in which the tournaments are conducted does not have a significant impact.

In conclusion, the order in which the tournaments are conducted should be chosen carefully to guarantee an improvement in generalization ability of the models, while still guaranteeing a good predictive performance.

5.3.2.2 Double Tournament Probability

The double tournament probability is another hyperparameter that can have a significant impact on the performance of the double tournament approach.

Starting with *DT*, which uses RMSE as the first objective and complexity as the second, it is possible to observe in Fig. 5.4 that in all datasets except for Concrete, reducing the double tournament probability leads to poorer generalization of the models. This trend shows that, with this tournament order, the double tournament probability acts as a selection pressure for simpler models, which translates in overfitting reduction. At a probability of 0.7 (*DT0.7*), the evolution produces models with better generalization compared to StdGP but not as good as *DT*, in most scenarios. Following the same trend, at a probability of 0.5 (*DT0.5*), the results resemble those of StdGP, indicating that this probability is too low and tends to produce overfitted models.

The Concrete dataset behaves differently, as StdGP already generalizes well, and using the Double Tournament selection tends to worsen overall performance, although without damaging generalization. In this dataset, reducing the double tournament probability leads to better model performance, indicating that *DT* is too extreme and unnecessarily penalizes models that are not overfitting.

On the other hand, when the reversed tournament order is used, the results are quite different. Fig. 5.5 shows the results of *InvDT* variants, which use complexity as the first objective and fitness as the second. As explained in section 5.3.2.1, when the double tournament probability is set to 1, *InvDT* is too extreme, leading to underfitting. Lowering this probability results in more models being chosen based solely on fitness, which reduces the over-penalization seen with probability 1. Interestingly, *InvDT0.5* seems to yield results similar to *DT*. This suggests that 0.5 is a probability that, in most cases, guarantees a good trade-of between complexity penalization and maintaining predictive ability in this inverted version of the Double Tournament.

All these methods were compared using the Wilcoxon signed-rank test, and the results are shown in Table 5.12. The highlighted values indicate statistical significance at the 0.05 level. The p-values indicate that *DT* effectively reduces overfitting compared to StdGP. However, as the double tournament probability decreases, this effectiveness also decreases. For example, the generalization ability difference between *DT0.5* and StdGP is only statistically significant on the Pollution and Bioavailability datasets, whereas *DT* shows statistically significant generalization across all datasets except BHousing.

Conversely, *InvDT* consistently shows significant generalization abilities compared to StdGP, regardless of the double tournament probability used. This suggests that *InvDT*is generally more effective at preventing overfitting than StdGPot *DT*, as this generalization ability is maintained with different double tournament probabilities. However, *InvDT* can lead to underfitting if the double tournament probability is too high, which is not desirable.

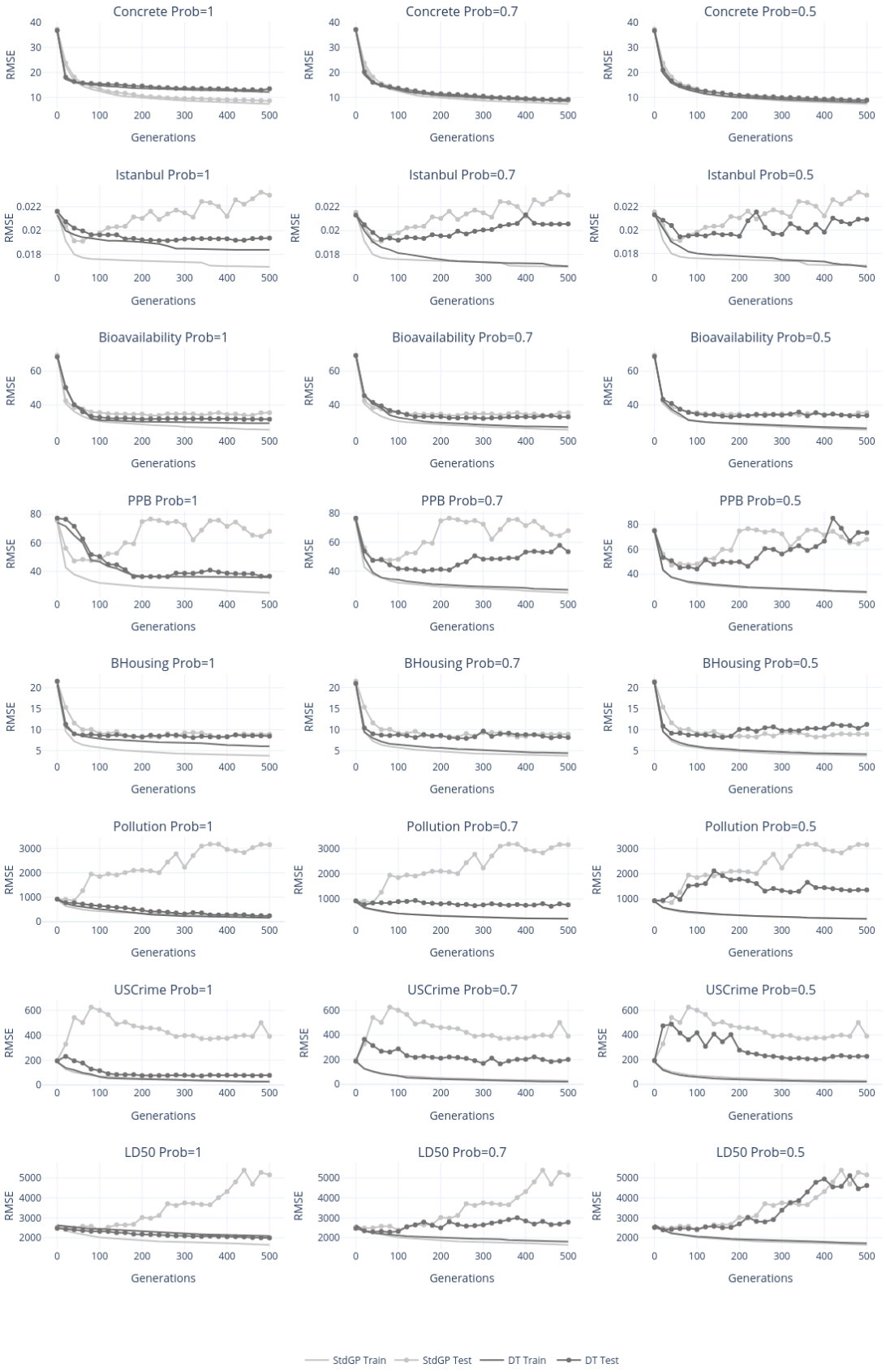


Figure 5.4: Median best-of-run performance using StdGP, *DT*, *DT0.7* and *DT0.5*. Each row represents a different dataset, and each column a probability.

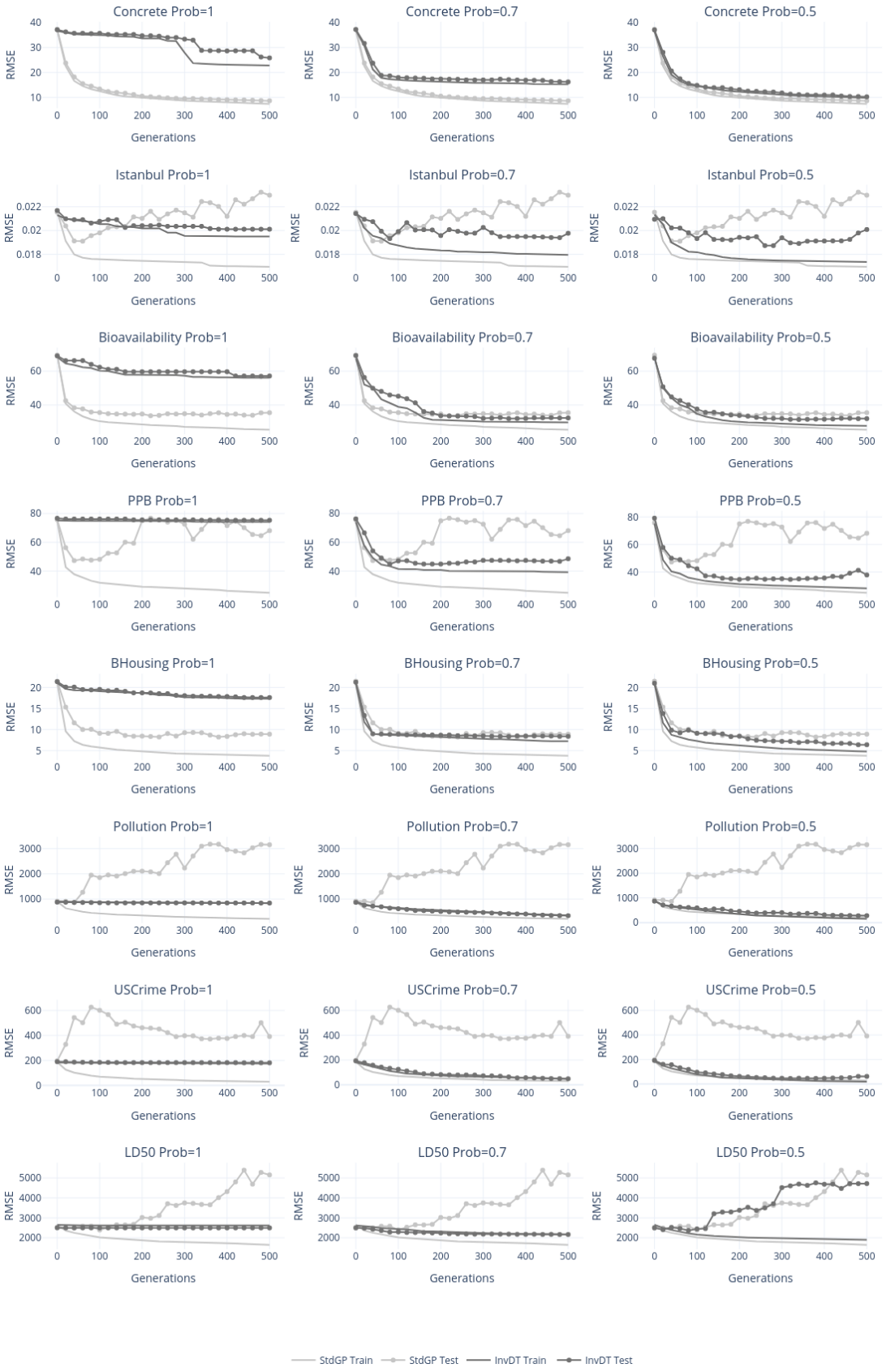


Figure 5.5: Median best-of-run performance using StdGP, *InvDT*, *InvDT0.7* and *InvDT0.5*. Each row represents a different dataset, and each column a probability.

Table 5.12: P-values of the Wilcoxon test for the difference between test and train error on the last generation in StdGP, *DT* and *InvDT* with probabilities 1, 0.7 and 0.5.

	USCrime	Pollution	Concrete	Istanbul	BHousing	PPB	LD50	Bioavailability
StdGP vs *DT*	**0.000**	**0.000**	**0.022**	**0.000**	0.096	**0.000**	**0.000**	**0.000**
StdGP vs *DT0.7*	**0.001**	**0.043**	**0.011**	0.339	0.777	0.146	0.096	**0.033**
StdGP vs *DT0.5*	0.280	**0.029**	0.129	0.073	0.221	0.428	0.393	**0.045**
StdGP vs *InvDT*	**0.000**	**0.000**	0.968	**0.000**	0.096	**0.000**	**0.000**	**0.001**
StdGP vs *InvDT0.7*	**0.000**	**0.000**	0.0.968	**0.001**	**0.025**	**0.000**	**0.000**	**0.001**
StdGP vs *InvDT0.5*	**0.000**	**0.000**	**0.016**	**0.026**	0.198	**0.004**	0.952	**0.000**
DT vs *DT0.7*	0.064	**0.001**	0.073	**0.006**	0.061	**0.003**	**0.000**	**0.000**
DT vs *DT0.5*	**0.015**	**0.000**	0.061	**0.002**	**0.011**	**0.001**	**0.000**	**0.000**
DT0.7 vs *DT0.5*	0.213	0.205	0.221	0.584	0.084	0.612	0.135	0.416
InvDT vs InvDT0.7	0.641	0.871	0.416	0.058	**0.007**	**0.029**	**0.006**	0.221
InvDT vs InvDT0.5	**0.001**	**0.007**	0.171	**0.002**	**0.016**	**0.000**	**0.000**	0.067
InvDT0.7 vs *InvDT0.5*	**0.011**	**0.001**	**0.028**	0.700	**0.016**	**0.000**	**0.000**	0.092
InvDT vs DT	**0.005**	0.058	0.067	0.191	**0.000**	**0.008**	0.164	0.903
InvDT vs DT0.7	**0.000**	**0.000**	0.670	**0.002**	**0.000**	**0.000**	**0.000**	**0.026**
InvDT vs DT0.5	**0.000**	**0.000**	0.935	**0.002**	**0.000**	**0.000**	**0.000**	0.109
InvDT0.7 vs *DT*	0.055	0.129	0.061	0.191	0.146	0.746	**0.028**	0.140
InvDT0.7 vs *DT0.7*	**0.000**	**0.000**	0.0318	**0.026**	**0.000**	**0.011**	**0.000**	0.052
InvDT0.7 vs *DT0.5*	**0.000**	**0.000**	0.465	**0.006**	**0.003**	**0.000**	**0.000**	**0.003**
InvDT0.5 vs *DT*	0.626	0.140	0.164	0.119	0.612	0.416	**0.000**	**0.000**
InvDT0.5 vs *DT0.7*	**0.000**	**0.033**	0.124	0.570	0.146	**0.047**	0.253	0.299
InvDT0.5 vs *DT0.5*	**0.000**	**0.000**	0.114	0.184	**0.040**	**0.022**	0.543	0.021

Finally, another interesting observation is that *DT* and *InvDT0.5* have similar generalization abilities in six out of eight datasets, indicating they are mostly equivalent in the ability to reduce overfitting. Both these versions also maintain a strong predictive performance in train and test sets, making them good choices to evolve robust models.

5.3.2.3 Tournament Sizes

Several combinations of tournament sizes were tested, and the results are shown in Figures 5.6 and 5.7. These results are from *DT* variants different tournament sizes. All other hyperparameters were kept fixed, as explained in section 5.2.2.1.

A combination that helps in avoiding overfitting, but has bad train and test predictive performance in most datasets is the one with 2 individuals in the first tournament and 4 in the second (*DT_2_4*). One possible explanation is that, larger tournaments increase selection pressure, making it more likely to choose the best individual overall. In this case, the fitness-based tournament is small while the complexity-based one is large, leading to low pressure in terms of fitness and high pressure in terms of complexity. This results in individuals that are bad in fitness and are overly penalized.

On the other hand, the combination that performs the worst in terms of generalization involves a first tournament size of 10 and a second tournament size of 2 (*DT_10_2*). Following the same reasoning as before, this pairing is too extreme in terms of fitness selection and too moderate in terms of complexity penalization, leading to high fitness pressure but low complexity pressure. Consequently, individuals selected with this combination outperform in fitness but are overly complex, resulting in overfitting.

In summary, the combination with the smallest fitness tournament size and largest complexity tournament size has poor predictive performance but succeeds in generalization. Conversely, the combination with the largest fitness tournament size and smallest complexity tournament size shows the best predictive performance during training but scores poorly in generalization. This contrast is interesting, and the results are coherent with each other.

The *DT_10_4* variant seems to have a good generalization across all datasets while maintaining strong predictive performance. Increasing the complexity tournament size from 2 to 4 when compared to *DT_10_2*, results in a middle ground that penalizes models enough to reduce overfitting without compromising predictive performance significantly. This balance is crucial for achieving good generalization and accuracy.

Regarding other combinations, it is possible to observe that versions with equal sizes for the first and second tournaments (*DT_2_2* and *DT_4_4*) also show somewhat weak predictive performance in most datasets. However, they succeed in avoiding overfitting, as indicated by the closeness of the train and test lines over generations. This suggests that a balanced selection pressure between the two objectives is not optimal to achieve good predictive performance, but is effective in overfitting avoidance.

The Wilcoxon signed-rank test was used to compare the different combinations of tournament sizes. Given the large number of comparisons and datasets, the median p-values across all datasets were calculated for each pair of methods, and the results are shown in Table 5.13. Highlighted values indicate statistical significance at the 0.05 level.

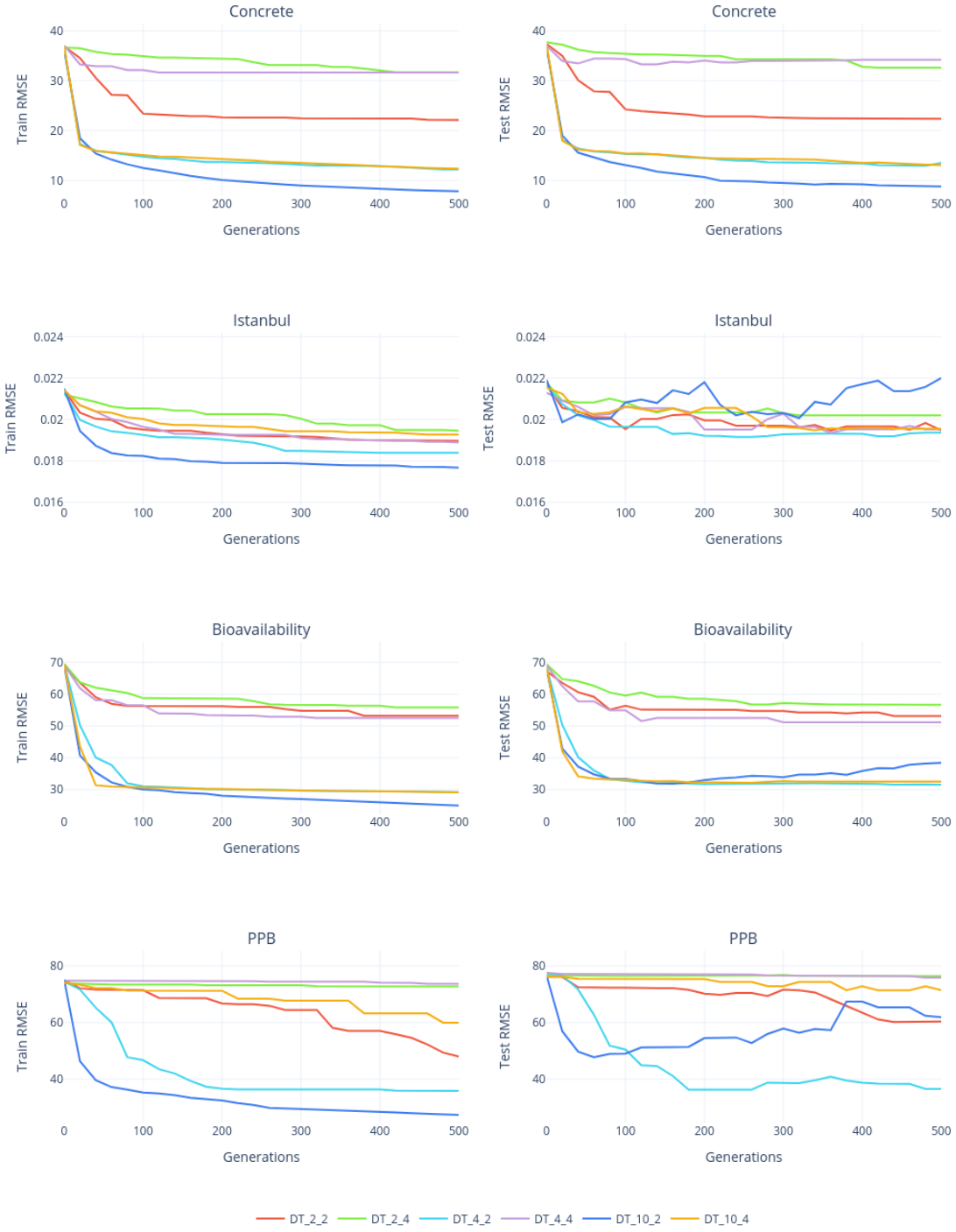


Figure 5.6: Median best-of-run performance throughout generations in datasets Concrete, Istanbul, Bioavailability and PPB, for *DT_2_2* (red); *DT_2_4* (green); *DT_4_2* (light blue); *DT_4_4* (purple); *DT_10_2* (dark blue); *DT_10_4* (orange). The left column and right columns show the train and test errors, respectively.

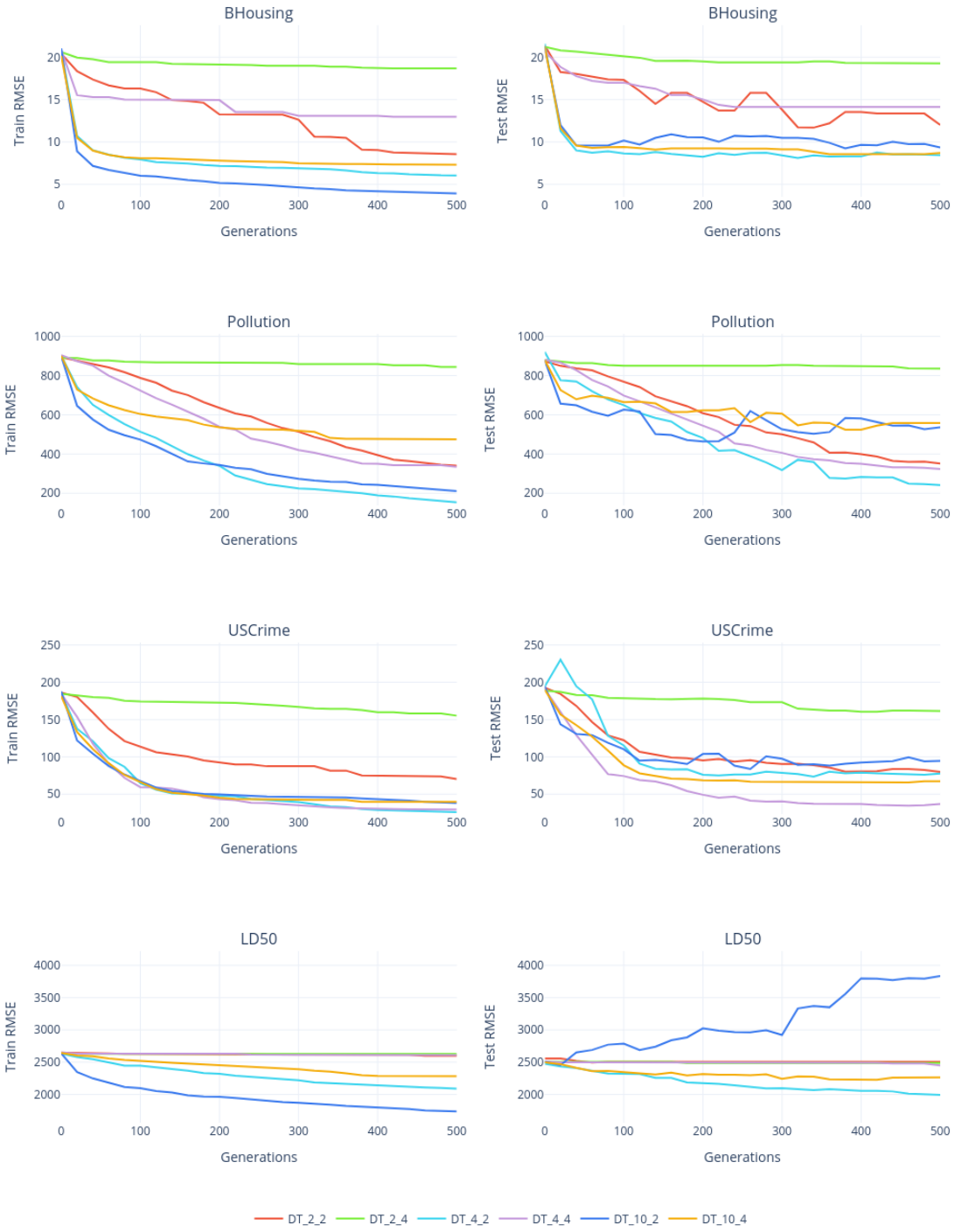


Figure 5.7: Median best-of-run performance throughout generations in datasets BHousing, Pollution, USCrime and LD50, for *DT_2_2* (red); *DT_2_4* (green); *DT_4_2* (light blue); *DT_4_4* (purple); *DT_10_2* (dark blue); *DT_10_4* (orange). The left column and right columns show the train and test errors, respectively.

Table 5.13: P-values of the Wilcoxon test for the difference between test and train error on the last generation in StdGP and *DT* with different combinations of tournament sizes. Each cell has the median p-value among the 8 datasets.

	DT_2_2	*DT_2_4*	*DT_4_2*	*DT_4_4*	*DT_10_2*	*DT_10_4*
StdGP	**0.000**	**0.000**	**0.000**	**0.000**	0.165	**0.000**
DT_2_2	-	0.351	0.217	0.310	**0.000**	0.209
DT_2_4	-	-	0.272	0.484	**0.005**	0.217
DT_4_2	-	-	-	**0.026**	**0.008**	0.412
DT_4_4	-	-	-	-	**0.000**	0.153
DT_10_2	-	-	-	-	-	**0.006**

All combinations are statistical significant in terms of generalization ability compared to StdGP, except for the *DT_10_2* variant. This variant is also significantly different from all other tested combinations, aligning with what was observed in Figures 5.6 and 5.7. The other variants are mainly not significantly different from each other, which indicates that, among these, the choice of tournament sizes does not have a significant impact on the generalization ability of the evolved models. Nevertheless, and once again, it is important to consider the general train and test performance of the models, as the choice of tournament sizes can have a significant impact on the predictive ability.

5.3.3 Evolved functions analysis

5.3.3.1 Curvature

After exploring the impact of various methods on reducing overfitting in GP, as well as the impact of different hyperparameters on the *DT* variant, it is interesting to analyze the evolved models in more detail. One way to do this is by looking at the curvature of the evolved functions, providing insights into the impact of penalizing complexity and how these functions compare to the target function.

Since both *DT* and *MMOTS* achieve very similar and competitive results in terms of generalization and predictive performance, models evolved by either method would be equally suitable for this analysis. Therefore, and for no particular reason, models evolved using the *DT* variant were chosen for the analysis. These models were compared to the ones evolved using StdGP, which does not penalize complexity during evolution.

An approximation of the curvature of the evolved functions can be obtained using the Slope-Based Complexity measure, which was used as a minimization objective in the *DT* variant. It is expected that the models evolved using *DT* have lower complexity

than those evolved using StdGP, as the former method penalizes overly complex models. This lower complexity should result in a lower curvature of the evolved functions, indicating that the models are smoother and less prone to overfitting.

The Slope-Based Complexity is calculated by adding together the partial complexities in each dimension of the function, which includes every feature used by the function. These partial complexities, formed by successive interconnected slopes, can be graphically represented in two-dimensional plots, with one plot for each feature, to observe the evolution of the curvature of the best individual throughout generations in each dimension. Given one partial complexity is calculated per dataset dimension, datasets with fewer features were chosen to demonstrate this analysis.

The Istanbul dataset, containing 7 features, was selected as the initial dataset. The approximations of the best-of-run function curvatures for each dataset dimension on generations 10, 150, and 500 are displayed in Fig. 5.8. For simplicity, the best-of-run individual was obtained from one random run out of the 30 GP runs performed. Each graph shows the feature values on the x-axis and the corresponding outputs on the y-axis for every observation in the dataset. These plots illustrate how the slopes between output values of observations contribute to calculating the partial complexities, and therefore, approximate of the function's curvature in each dimension. This visual representation resembles Figure 4.4(b), demonstrating the computation of slopes.

The figure shows that the curvature of the evolved models using *DT* consistently remains lower than those evolved using StdGP. Although there are still plenty of oscillation in the 150th and last generations, these are always less pronounced when compared to the ones of StdGP. This indicates the models evolved with *DT* are smoother, directly reflecting their lower complexity and reduced tendency to fit noise in data. This lower complexity is a result of the penalty imposed by *DT* on overly complex models, which is not present in StdGP.

In contrast, Fig. 5.9 illustrates another random run of the GP evolution using the Istanbul dataset. In this run, the curvatures on the features used by the best-of-run individual in both StdGP and *DT* at the 150th and 500th generation seem to have a very similar degree of curvature. Both functions show significant oscillation with similar amplitude and frequencies. However, the best-of-run individual evolved with *DT* uses only 4 out of the 7 dataset features, whereas the best-of-run individual evolved with StdGP uses all 7 features. As the SBC is computed as the sum of partial complexities over each feature used by the function, a function that uses fewer features always has lower complexity.

This is an interesting observation, as it highlights the two main benefits of using the Slope-Based Complexity measure to approximate the complexity of functions. On one hand, minimizing this measure results in smoother models, as seen in Fig. 5.8. On the other hand, it also promotes feature selection, as seen in Fig. 5.9. This embedded feature selection process is essential in reducing overfitting, as it prevents the models from using features that only enhance performance on the training set but not on the

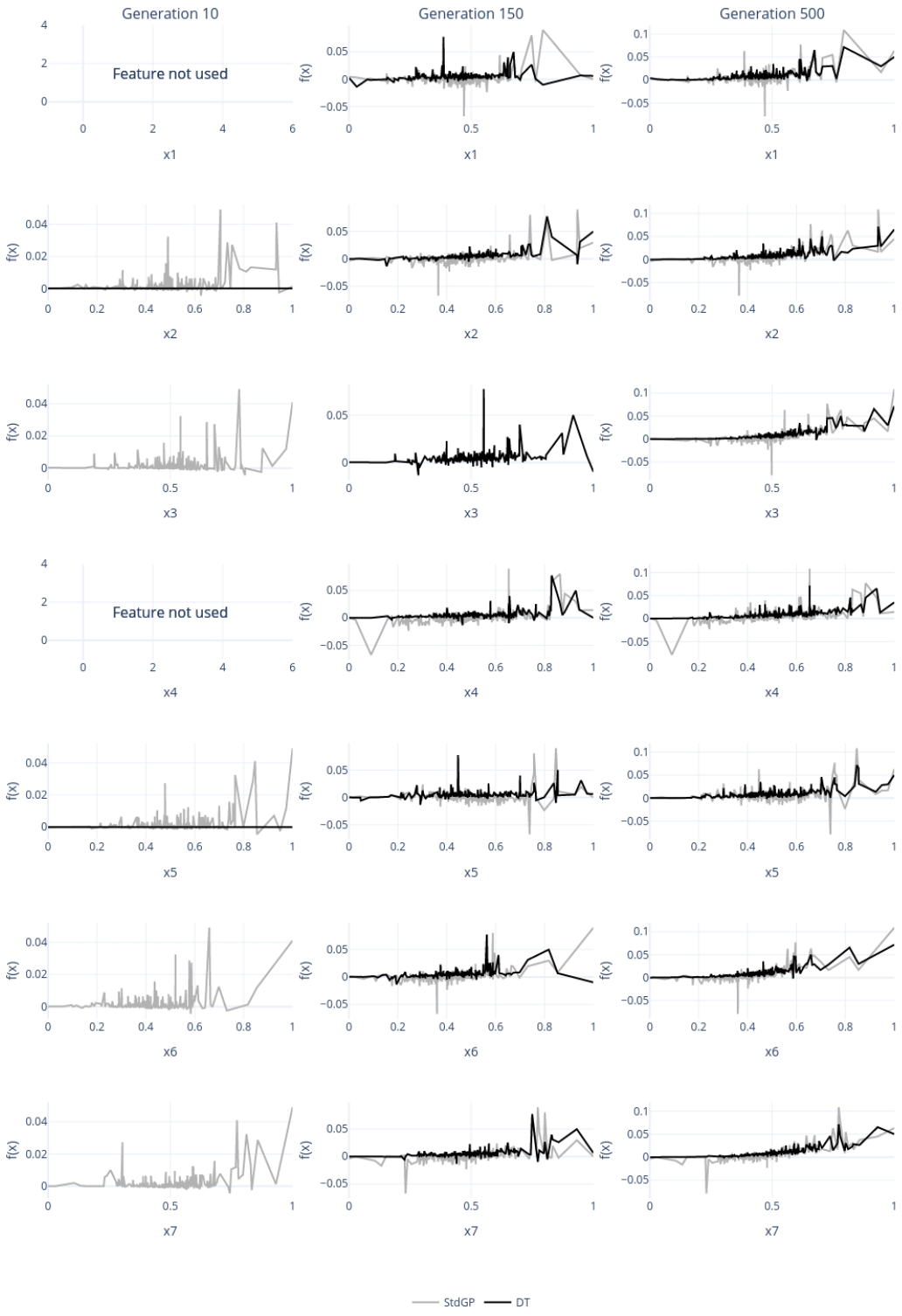


Figure 5.8: Approximation of the curvature of the best-of-run using SBC's partial complexity. Each row represents an approximation of the curvature of the function in one of the 7 dimensions of the Istanbul dataset, and each column a generation. The best-of-run was obtained from one random GP run.

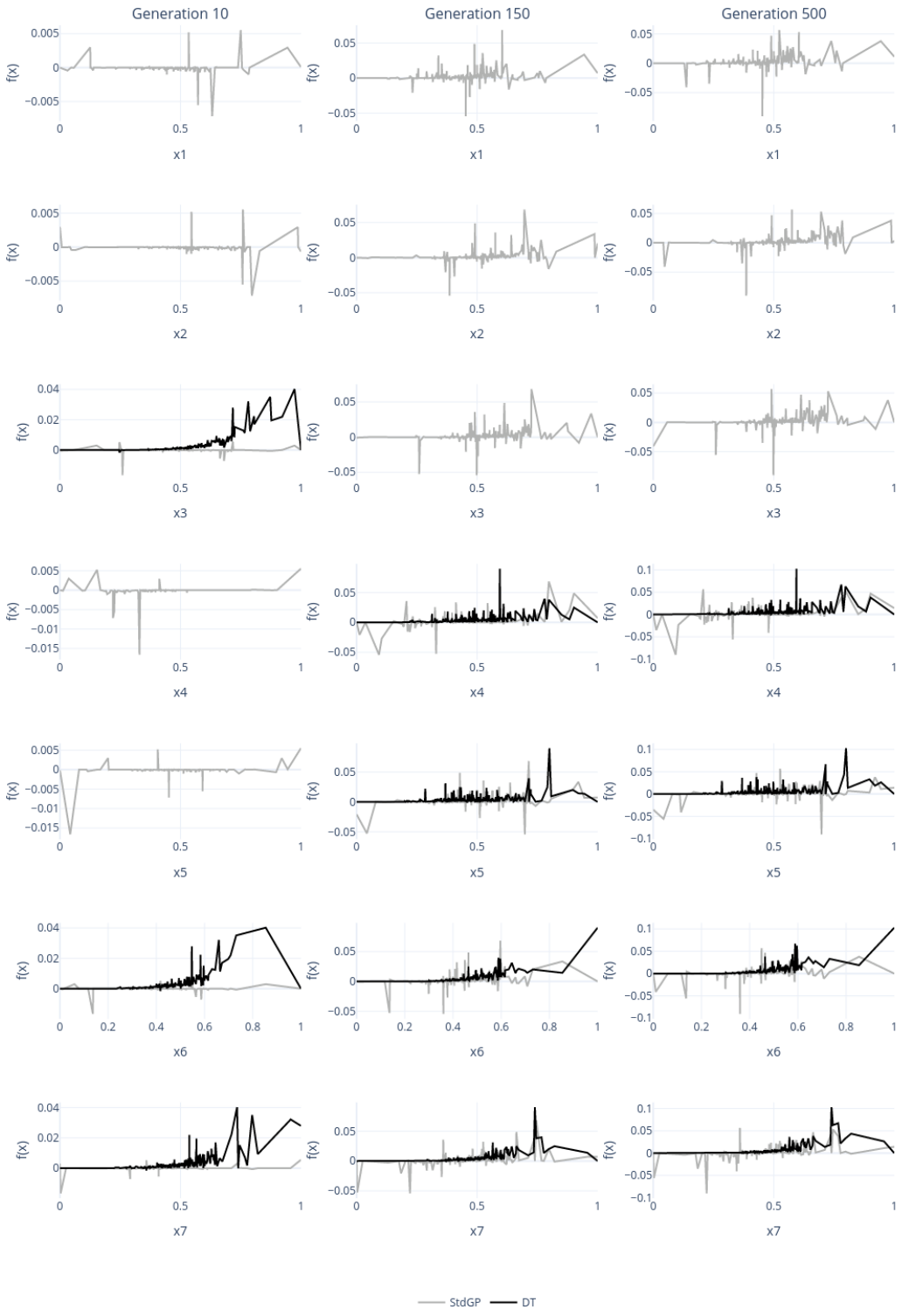


Figure 5.9: Approximation of the curvature of the best-of-run using SBC's partial complexity. Each row represents an approximation of the curvature of the function in one of the 7 dimensions of the Istanbul dataset over the generations. The best-of-run was obtained from another random GP run.

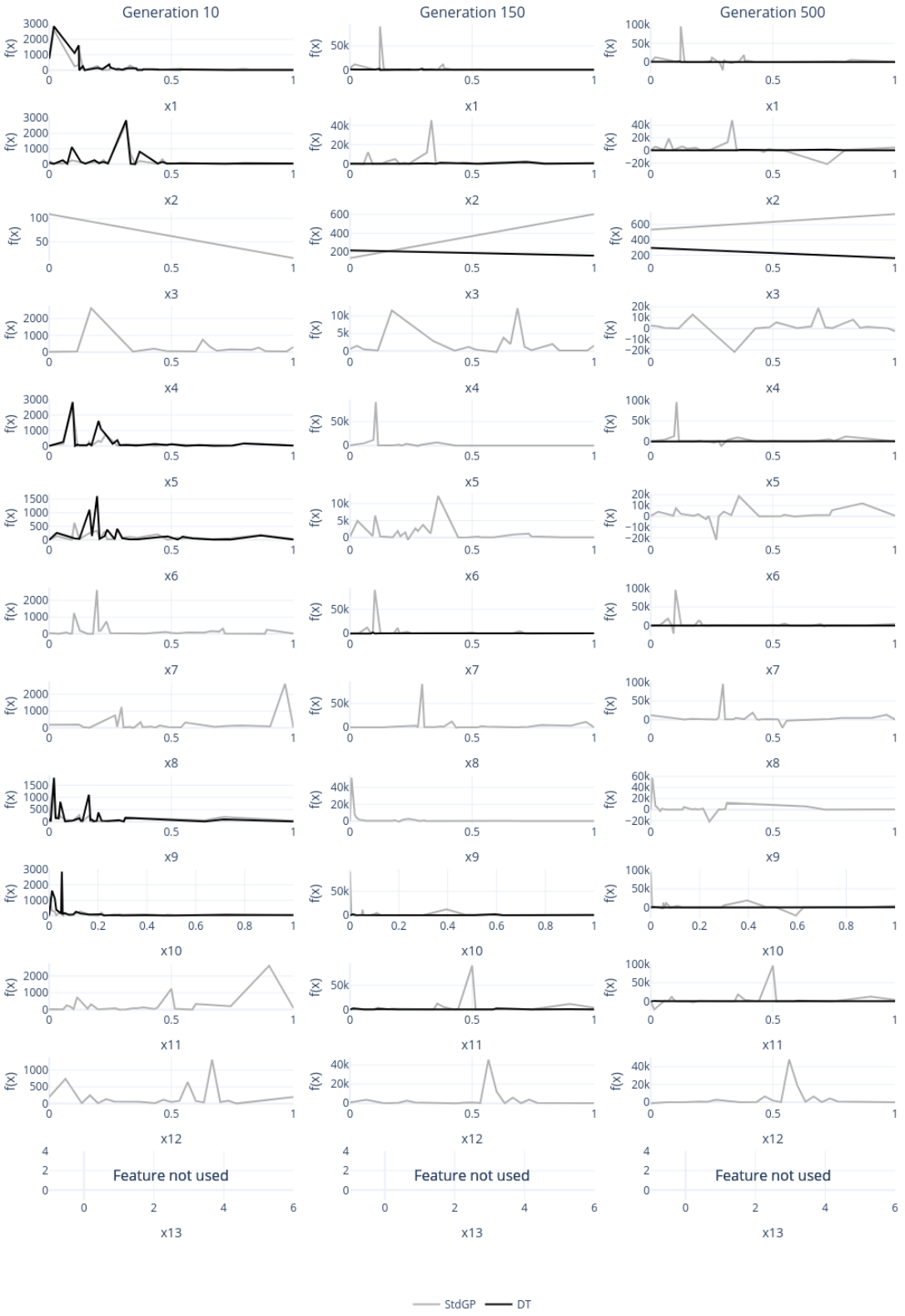


Figure 5.10: Approximation of the curvature of the best-of-run function using SBC. Each row represents an approximation of the curvature of the function in one of the 13 dimensions of the USCrime dataset, over the generations. The best-of-run was obtained from one random GP run.

test set.

Another example of these SBC benefits and impacts on the evolved models can be seen in Fig. 5.10. This figure shows the approximation of the curvature of the best-of-run individual throughout generations in a random GP run using the USCrime dataset, which contains 13 features. Here, it is evident that minimizing SBC not only results in smoother functions but also in a model that uses only 7 out of the 12 features used by the best-of-run evolved by StdGP. In fact, the best-of-run individual evolved using *DT* is so much smoother than the one evolved using StdGP that the oscillations are barely visible in the 150th and 500th generations due to the scale of the plot. This is a clear indication of the effectiveness of the selection algorithms that minimize the Slope-Based Complexity measure in promoting feature selection and reducing overfitting.

5.3.3.2 Feature Importance

The SBC measure is not only useful for approximating the curvature of the evolved functions but also for quantifying each feature's contribution to the complexity of the function and promoting feature selection. This is an important aspect of the complexity measure, as it helps in reducing overfitting by preventing the models from using features that only enhance performance on the train set but not on the test set.

SBC is calculated by adding up the partial complexities across each dimension of a function, which includes all the features used by the function. As a consequence, if a function uses fewer features, its complexity automatically decreases. This encourages feature selection, pushing models to use only the most relevant features. Moreover, this method helps quantify the importance of each feature in the evolved models and enables the analysis of how much each feature contributes to the function's complexity.

Fig. 5.11 shows the contribution of each feature to the complexity of the best-of-run individuals evolved using StdGP, *DT* and *MMOTS* in generations 10, 150, and 500. The feature contribution is given by the partial complexity of that feature on the SBC measure. The results show the median partial complexities across the 30 GP runs performed on the datasets with a manageable number of features: Istanbul, Concrete, BHousing and USCrime, containing 7, 8, 13 and 13 features, respectively The logarithm of the partial complexities was used to better visualize the results, as the absolute values of partial complexity can be very high.

The results consistently show that the partial complexities in the best-of-run individuals evolved using the complexity minimization variants *DT* and *MMOTS* are lower compared to those of functions evolved using StdGP. This suggests that *DT* and *MMOTS*-evolved functions use features in a more linear and less complex manner, resulting in smoother functions. This outcome is a direct consequence of the SBC minimization throughout the evolutionary process.

Given the increasing interest in model interpretability within the machine learning community, these results are particularly relevant as they offer insights into the

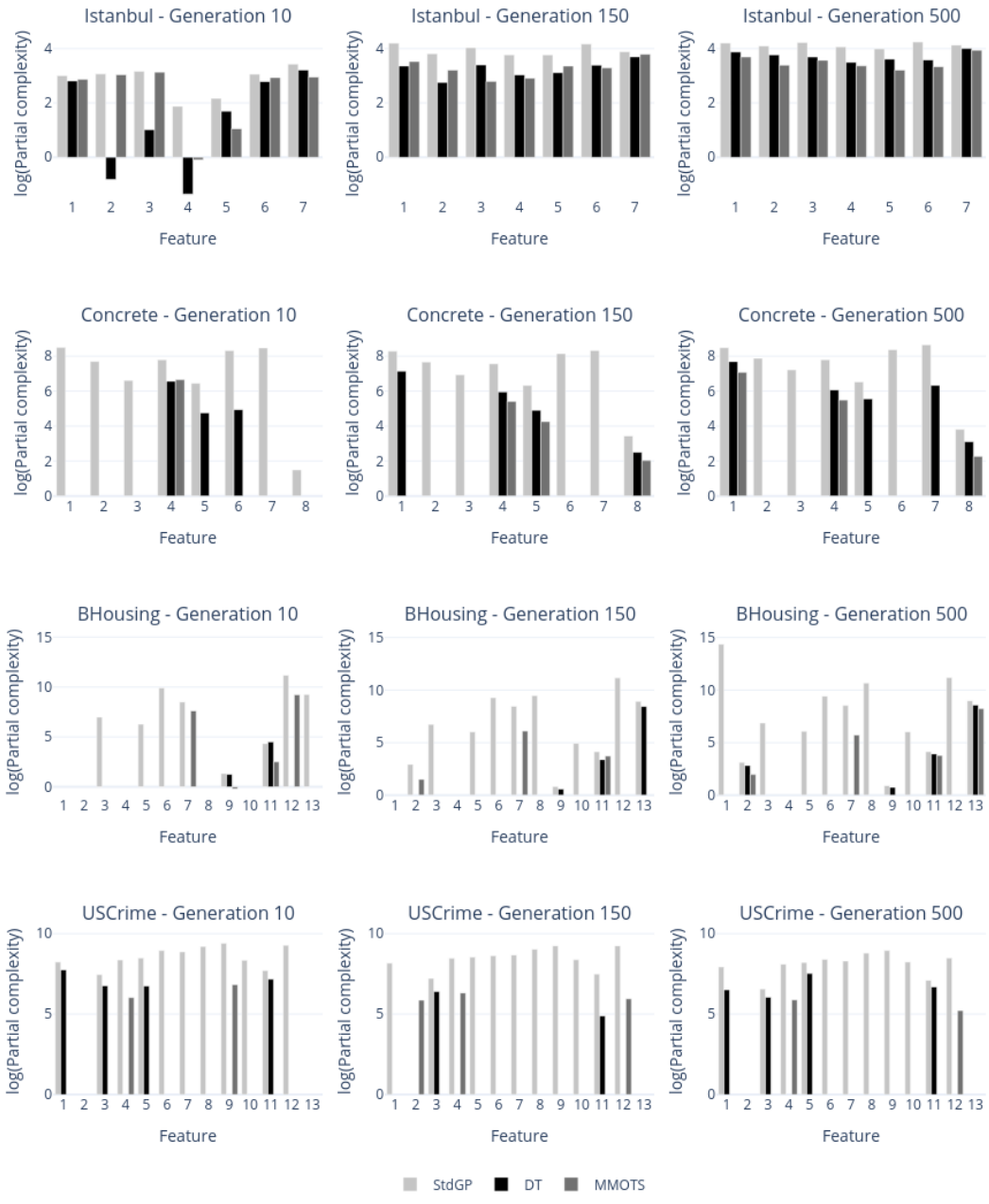


Figure 5.11: Partial complexities of the best-of-run individuals evolved using StdGP (light gray), *DT* (black) and *MMOTS* (dark gray) in generations 10, 150 and 500.

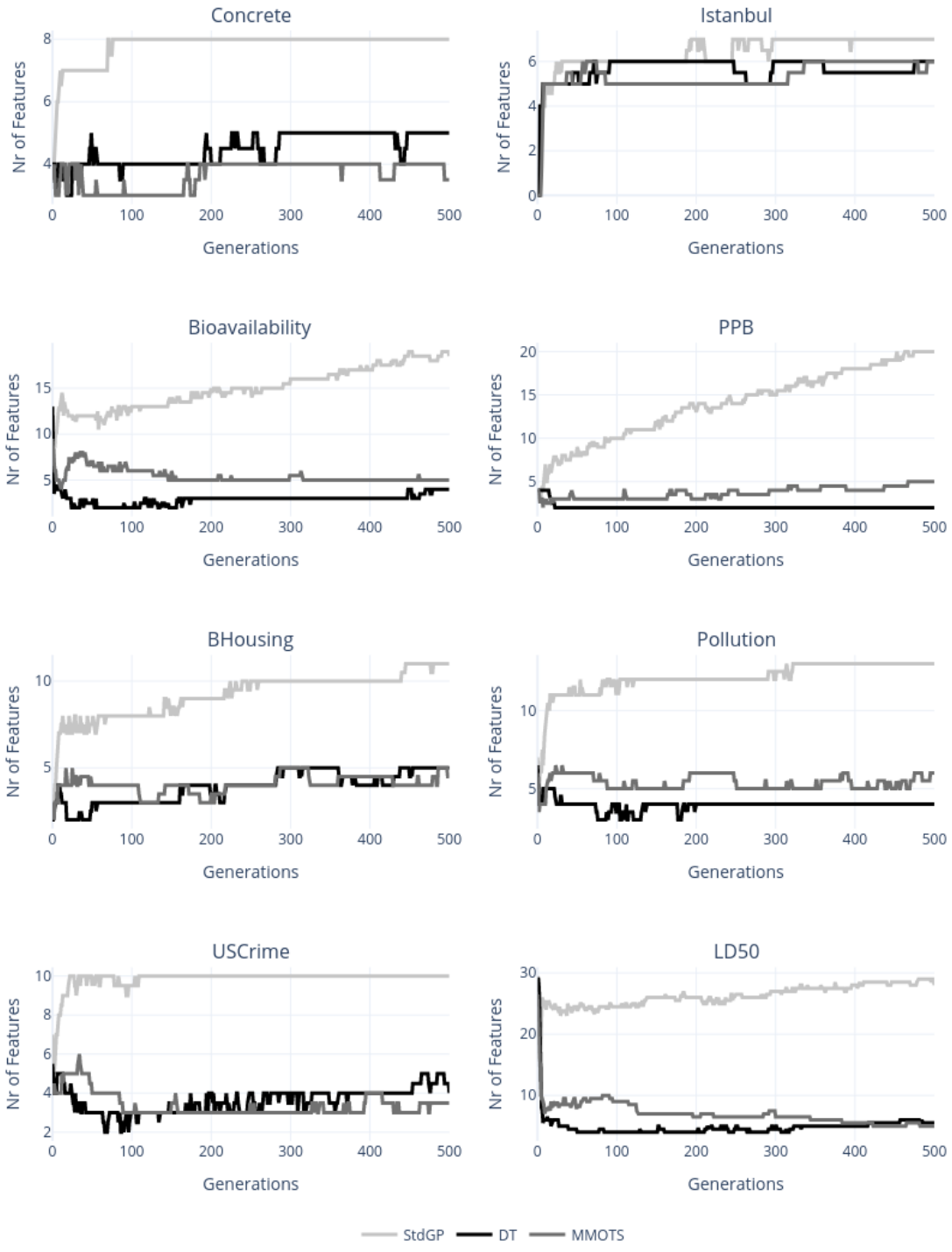


Figure 5.12: Number of features used by the best-of-run individuals in StdGP (light gray), *DT* (black) and *MMOTS* (dark gray) throughout the generations.

importance of each feature in the evolved models.

Moreover, it is evident that the best-of-run individuals evolved using *DT* and *MMOTS* tend to use fewer features compared to the StdGP-evolved ones. This shows how feature selection is promoted by the minimization of SBC, which plays a crucial role in mitigating overfitting. These observations are further supported by Fig. 5.12, which shows the number of features used by the best-of-run individuals evolved using all GP variants throughout the generations. The results show the median number of features used across the 30 GP runs performed.

Once again, the results confirm that individuals evolved using complexity minimization strategies tend to use fewer features compared to those evolved using StdGP. These findings are important as they show the effectiveness of the Slope-Based Complexity measure in encouraging feature selection and quantifying each feature's contribution to function complexity.

Regarding the difference in feature selection effectiveness between *DT* and *MMOTS*, the results demonstrate that both methods effectively promote feature selection. At times, one method evolves models with fewer features than the other, and vice versa. This indicates that both *DT* and *MMOTS* are effective in encouraging feature selection and reducing overfitting, with no clear preference for one method over the other in this aspect.

5.3.3.3 Model Complexity and Overfitting

The methodology proposed in this work is based on the assumption that overfitting can be reduced by penalizing overly complex models. The results from previous sections have shown that the proposed strategies for minimizing complexity, specifically *DT* and *MMOTS*, effectively reduce overfitting, supporting this assumption.

Figure 5.13: Best-of-Run Slope-Based Complexity throughout the generations in StdGP (light gray), *DT* (black) and MMOTS (dark gray).

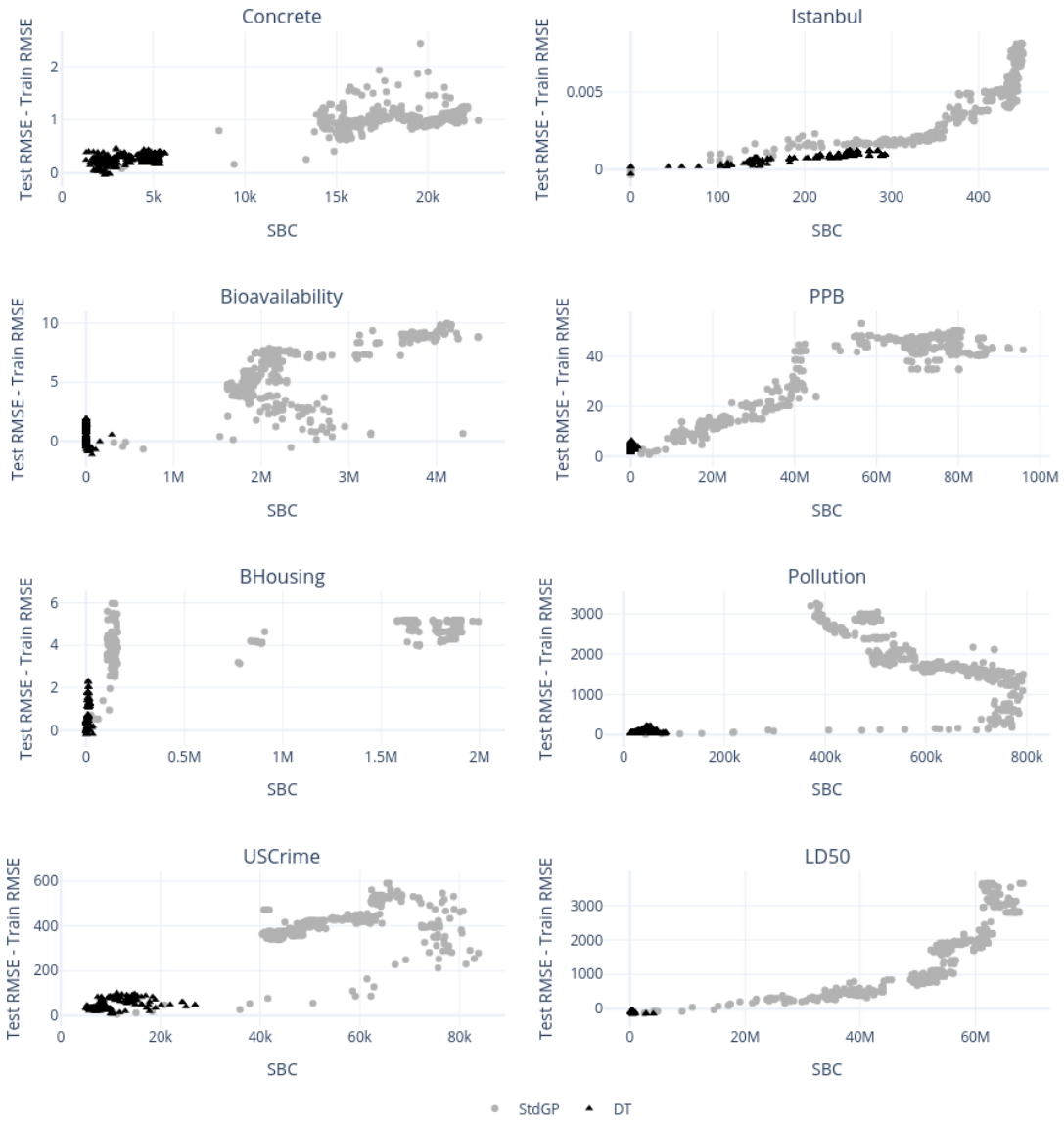


Figure 5.14: Best-of-Run SBC vs Overfitting.

However, it is important to analyze how the complexity of the best-of-run individual evolves throughout generations to ensure that the complexity minimization strategies are indeed reducing complexity, and that this reduction is correlated with the reduction in overfitting. Therefore, the complexities of the best-of-run individuals using StdGP, *DT*, and *MMOTS* were calculated throughout the generations, and the results are displayed in Fig. 5.13. The SBC values were averaged across the 30 GP runs performed on each dataset and are plotted using the logarithmic scale to better visualize the results, given the absolute values of SBC can be very high.

The figure shows that the complexity of the best-of-run individual evolves differently in each approach. With StdGP, the complexity consistently increases throughout the generations in most datasets, which indicates that the models are becoming more complex as the evolution progresses. Even when complexity stops increasing, its value remains high. This results in overfitting, as overly complex models are more likely to fit noise in the training data, which results in poor generalization ability. Conversely, the complexity of the best-of-run individual using *DT* and *MMOTS* remain relatively stable and low. This indicates that the complexity minimization strategies are effective in penalizing overly complex models.

It is now time to look at the correlation between complexity and overfitting more closely. Fig. 5.14 shows the variation in the difference between test and train error, serving as a proxy for overfitting, in relation to the complexity of the best-of-run individual. For this visualization, the best-of-run individuals in every generation of both StdGP and *DT* were used to guarantee the presence of individuals with both low and high complexities.

In most datasets, one can observe a clear positive correlation between complexity and overfitting. As complexity increases, the difference between test and train error also increases, indicating that overly complex models are more likely to overfit the training data. This correlation is particularly evident in the Istanbul, PPB and LD50 datasets, where the difference between test and train error is consistently higher for models with higher complexity. On the contrary, the points belonging to the best-of-run individuals evolved using *DT* are mostly clustered in the lower left corner of the plots, indicating that these models have lower complexity and also overfit less.

Now that an intuitive correlation between complexity and overfitting is established, this correlation can be formally established using the Spearman's rank correlation coefficient. This measure is used to assess the strength and direction of the relationship between two ranked variables, in this case, the complexity of the best-of-run individual and the generalization ability of the models. The correlation coefficients can be found in Table 5.14.

The high correlation coefficients indicate a strong positive correlation between complexity and overfitting in most datasets. This confirms the intuition that overly complex models are more likely to overfit the training data, while simpler models are more likely to generalize well. This analysis provides further evidence that complexity

minimization is an effective strategy for reducing overfitting in GP.

Table 5.14: SBC and overfitting Spearman correlation coefficients.

Dataset	Correlation
Concrete	0.831
Istanbul	0.924
Bioavailability	0.866
PPB	0.894
BHousing	0.906
Pollution	0.723
USCrime	0.843
LD50	0.839

5.3.3.4 Model Size

Given the importance of model size and bloat control in GP, it is worth analyzing the impact of reducing overfitting on the size of the evolved models. Model size refers to the number of nodes in the evolved tree, a common measure in the GP literature.

Furthermore, given that several previous studies focused on penalizing larger models as an attempt to reduce overfitting, it is interesting to explore if there is a link between reducing overfitting by limiting the functional complexity of models, and the resulting size of those models. This analysis can provide insights into the relationship between model size and complexity, and how complexity minimization techniques impact the size of evolved models.

Fig. 5.15 shows that, in some datasets, *DT* evolves smaller models when compared to StdGP. However, it does not verify for 3 datasets: Pollution, USCrime and LD50. This indicates that, although penalizing complexity can lead to smaller models, this is not always the case. This is an interesting observation, as it suggests that the relationship between model size and complexity is not always straightforward, and that other factors may influence the size of the evolved models. Similarly, *MMOTS* does not always evolve smaller models than StdGP, as visible in the PPB, Pollution and USCrime datasets.

When comparing *DT* with *MMOTS*, the benefit of one over the other is also not clear. In some datasets, *DT* evolves smaller models, while in others, *MMOTS* does. Nevertheless, *MMOTS* generally evolves smaller models than *DT*. Despite that, the difference in size is not very significant in most datasets and the choice between the two methods should not be based on the size of the evolved models.

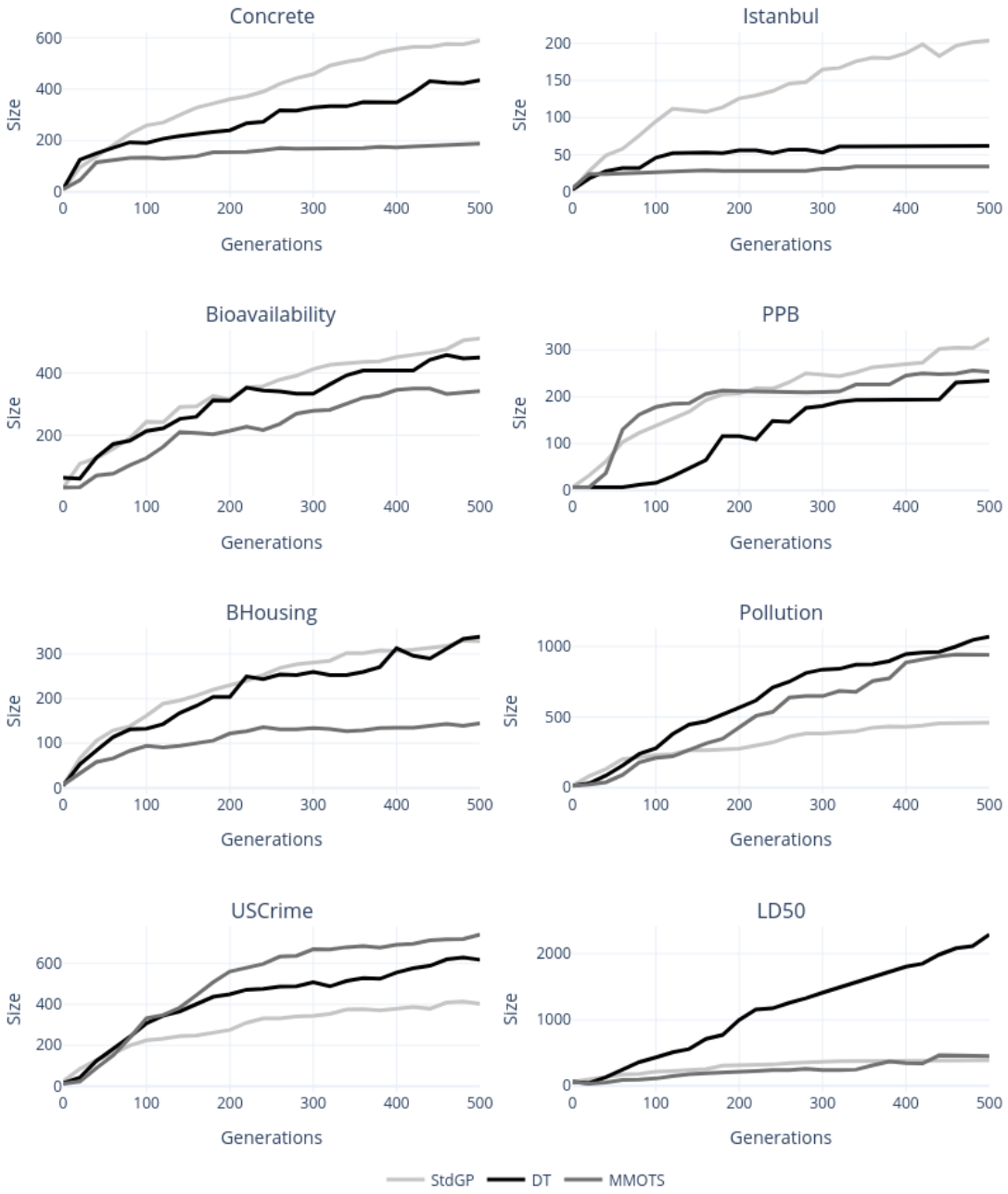


Figure 5.15: Best-of-run size evolution throughout the generations. The results show the median values of 30 independent runs.

It is also possible to visualize the relationship between model size and complexity in a more straightforward plot. Although it is not a common visualization, the plots in Fig. 5.16 show how the size of the evolved best-of-run model varies with its complexity, measured by the Slope-Based Complexity. This visualization was obtained for models evolved using StdGP and *DT*, and the results are the median values of 30 independent runs. *MMOTS* was not included in this analysis for simplicity.

The figure shows that, for all datasets, limiting the complexity of the models almost never leads to a limitation in the size of the models. As it is shown by the evident almost vertical lines in the plots, while the complexity of the models is consistently lower when using *DT*, the best-of-run individuals still achieve a similar size to the ones evolved using StdGP. This indicates that penalizing overly complex models is effective in reducing overfitting without necessarily reducing the size of the evolved models.

The conclusions from this analysis align with those from earlier research, such as the study by Vanneschi et al. (2010). That study demonstrates that bloat, complexity, and overfitting do not have a direct correlation, and restricting bloat does not always result in reduced overfitting. This analysis shows that reducing overfitting by limiting the function complexity of evolved models also does not automatically address bloat issues.

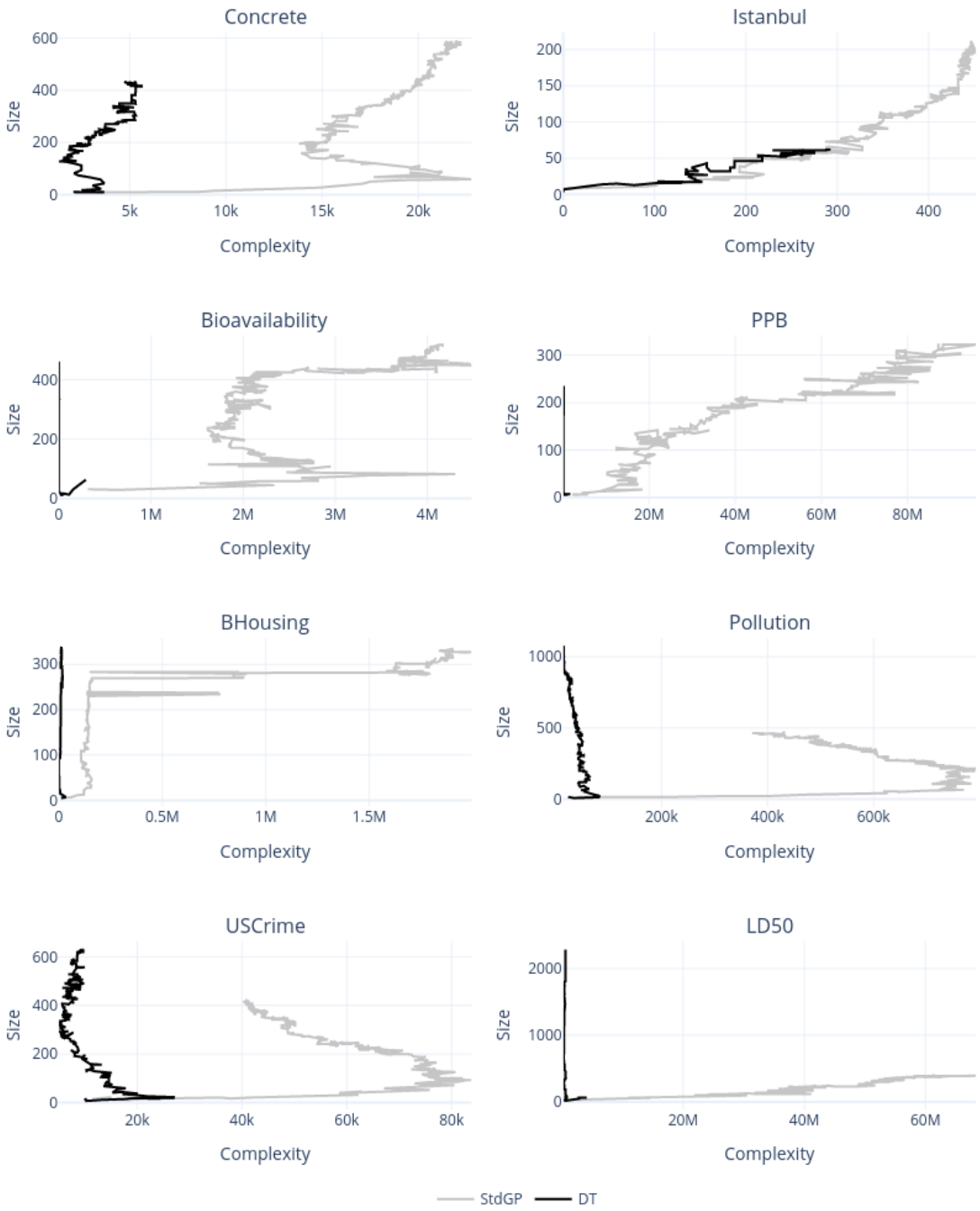


Figure 5.16: Best-of-Run complexity vs size. The results show the median values of 30 independent runs.

6

Conclusions and Future Work

To address the long-lasting challenge of overfitting in GP, this study introduces a new method focused on simplifying the functional complexity of evolved models. Functional complexity refers to the phenotype of a function, which reflects its behavior.

Overfitting, a common issue in ML techniques including GP, arises when a model becomes too adjusted to the training data, capturing noise and irrelevant patterns rather than the data's underlying structure. GP's ability to evolve diverse functions without constraints on their shape and size makes it a promising approach among other ML algorithms, but this flexibility also makes it prone to overfitting. GP has the potential to develop models that fit the training data exceptionally well. However, there is a tendency for these models to become excessively complex, which ultimately compromises their ability to generalize effectively.

The proposed approach aims to mitigate this issue by using a functional complexity measure called Slope-Based Complexity to approximate a function's curvature, and applying multi-objective selection algorithms to minimize both training error and complexity. The goal is to evolve models that are simpler and smoother, reducing the risk of overfitting to training data. Although these components are not new to GP, their combination represents a novel approach.

Two multi-objective selection schemes were evaluated: the Double Tournament and the Modified Multi-Objective Tournament Scheme. GP variants that use these selection methods, labeled as *DT* and *MMOTS*, respectively, aim to optimize both fitness and complexity, thereby evolving models that achieve both accuracy and generalizability. Comparing them to the StdGP, which employs a simple tournament selection, both *DT* and *MMOTS* effectively mitigate overfitting. These GP variants consistently generated models with reduced complexity and improved generalization compared to those produced by StdGP. Regarding the preference for one method over the other, no statistical difference was observed in terms of generalization performance in most datasets. Additionally, both methods also maintained a similar strong predictive performance across both training and testing datasets.

Further analysis of *DT*'s hyperparameters revealed that tournament order, double tournament probability, and tournament sizes significantly impact performance. When fitness is prioritized as the first tournament criterion and complexity as the second, reducing the double tournament probability led to poorer generalization performance. On the other hand, using the reversed tournament order consistently showed a significant overfitting reduction, with noticeable improvements in both training and testing predictive abilities observed as the probability decreased.

Regarding tournament sizes, results indicate that employing a larger fitness tournament size and a smaller complexity tournament size, while ensuring a balanced difference between them, optimally reduced overfitting and improved predictive performance. This configuration effectively balances the emphasis on fitness minimization without overly penalizing complexity.

In short, prioritizing fitness over complexity while using a double tournament probability of 1, and choosing a larger first tournament size and smaller second tournament size, ensuring that the difference between the two is not very high, yielded the best results in terms of overfitting avoidance and overall train and test performance. Alternatively, inverse order prioritization and lower double tournament probability also produced similar outcomes in most datasets.

A detailed analysis of the best-of-run individuals evolved using *DT* and *MMOTS* was conducted. Both variants effectively reduced model complexity, with individuals consistently exhibiting lower complexity compared to StdGP counterparts. Complexity remained stable and low across generations for *DT* and *MMOTS*-evolved models, contrasting with the increasing complexity in StdGP-evolved ones. This aligns with the observed positive correlation between complexity and overfitting, where overly complex models tend to overfit.

The analysis also concluded that *DT*-generated individuals are smoother and less complex compared to those achieved with StdGP. This was evident when looking into the shape of evolved function by visualizing the slopes that form the partial complexities of the best-of-run individuals for each dataset dimension, consistently showing lower curvature for *DT*-evolved individuals. This outcome is a direct consequence of the complexity minimization strategy employed by the proposed method. Although this analysis was only performed for *DT*-evolved models for simplicity, it is expected that *MMOTS*-evolved models would exhibit similar behavior, given their comparable performance.

DT and *MMOTS* also encourage models to use fewer features, which is crucial for reducing overfitting as fewer features lead to simpler models that fit less noise in training data. Results consistently showed that the best-of-run individual obtained with both complexity minimization variants use fewer features compared to those from StdGP. Even in a dataset where StdGP generalizes exceptionally well, *DT* and *MMOTS* discovered non-overfitting models that utilized fewer features. This demonstrates that

these variants are not only effective in reducing overfitting but also in encouraging feature selection, which enhances interpretability. Moreover, the analysis of each feature contribution to complexity not only provides interpretability regarding impact of each feature on the model's complexity, but also showed that the feature contribution to complexity is lower in *DT* and *MMOTS*-evolved models compared to StdGP-evolved ones.

Interestingly, while the best-of-run models obtained using the complexity minimization variants consistently showed lower complexity than StdGP ones, these were not always smaller. This suggests that penalizing complexity effectively reduces overfitting without necessarily reducing model size. This finding relates to the notion that mitigating bloat does not always reduce overfitting, and adds the idea that reducing complexity also does not always lead to bloat reduction.

In summary, the proposed GP variants effectively mitigate overfitting by minimizing functional complexity alongside using two different multi-objective selection schemes. Both *DT* and *MMOTS* consistently evolved models with reduced complexity and improved generalization compared to StdGP. These variants also maintained strong predictive performance across both training and testing datasets. No significant difference was observed between *DT* and *MMOTS* in terms of generalization performance, predictive performance, or bloat.

Both these methods have their own sets of advantages and disadvantages. For instance, *DT* has the advantage of being computationally efficient, while *MMOTS* is more computationally intensive due to the need to scale complexity and error for each individual in every generation. Additionally, *DT* offers greater flexibility and can be adjusted to various scenarios through parameter tuning, whereas *MMOTS* is more rigid and less adaptable. On the other hand, this flexibility in *DT* can be seen as disadvantage as it requires parameter tuning, while *MMOTS* is more straightforward to implement.

Future work could focus on examining how the suggested methods work on different types of problems like other symbolic regression benchmarks, classification tasks and other GP applications. It would also be interesting to explore the impact of the proposed approach using different functional complexity measures. Different multi-objective selection schemes could be tested to further investigate the impact of minimizing complexity on reducing overfitting in GP. Finally, comparing the proposed approaches with other methods that aim to reduce overfitting in GP would also be relevant.

Bibliography

Agapitos, A., Brabazon, A., & O'Neill, M. (2012). Controlling overfitting in symbolic regression based on a bias/variance error decomposition. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, & M. Pavone (Eds.), *Parallel problem solving from nature - ppsn xii* (pp. 438–447). Springer Berlin Heidelberg.

Akbilgic, O., Bozdogan, H., & Balaban, M. E. (2013). A novel hybrid rbf neural networks model as a forecaster. *Statistics and Computing*, *24*, 365–375. https://api.semanticscholar.org/CorpusID:17764829

Akkurt, S., Ozdemir, S., Tayfur, G., & Akyol, B. (2003). The use of ga-anns in the modelling of compressive strength of cement mortar. *Cement and Concrete Research*, *33*(7), 973–979. https://doi.org/https://doi.org/10.1016/S0008-8846(03)00006-1

Alonso, C. L., Montaña, J. L., & Borges, C. E. (2016). Genetic programming model regularization. In K. Madani, A. Dourado, A. Rosa, J. Filipe, & J. Kacprzyk (Eds.), *Computational intelligence* (pp. 105–120). Springer International Publishing.

Alpaydin, E. (2014). *Introduction to machine learning* (3rd ed.). MIT Press.

Archetti, F., Lanzeni, S., Messina, E., & Vanneschi, L. (2006). Genetic programming for human oral bioavailability of drugs. *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, 255–262. https://doi.org/10.1145/1143997.1144042

Archetti, F., Lanzeni, S., Messina, E., & Vanneschi, L. (2007). Genetic Programming and Other Machine Learning Approaches to Predict Median Oral Lethal Dose (LD50) and Plasma Protein Binding Levels (%PPB) of Drugs. In E. Marchiori, J. H. Moore, & J. C. Rajapakse (Eds.), *Evolutionary computation,machine learning and data mining in bioinformatics* (pp. 11–23). Springer Berlin Heidelberg.

Archetti, F., Lanzeni, S., Messina, V., & Vanneschi, L. (2007). Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines*, *8*, 413–432. https://doi.org/10.1007/s10710-007-9040-z

Azad, R. M. A., & Ryan, C. (2011). Variance based selection to improve test set performance in genetic programming. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 1315–1322. https://doi.org/10.1145/2001576.2001754

Banzhaf, W., Francone, F. D., & Nordin, P. (1996). The effect of extensive use of the mutation operator on generalization in genetic programming using sparse data sets. In H.-M. Voigt, W. Ebeling, I. Rechenberg, & H.-P. Schwefel (Eds.), *Parallel problem solving from nature — ppsn iv* (pp. 300–309). Springer Berlin Heidelberg.

Bhardwaj, B., Agarwal, D., & Srivastava, G. (2020). Theoretical framework of ensemble methods and applications. *IOSR Journal of Engineering*, *10*, 58–63.

Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Springer-Verlag.

Bomarito, G. F., Leser, P. E., Strauss, N. C. M., Garbrecht, K. M., & Hochhalter, J. D. (2022). Bayesian model selection for reducing bloat and overfitting in genetic programming for symbolic regression. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 526–529. https://doi.org/10.1145/3520304.3528899

Bourmistrova, A., & Khantsis, S. (2007). Control system design optimisation via genetic programming. *2007 IEEE Congress on Evolutionary Computation*, 1993–2000. https://doi.org/10.1109/CEC.2007.4424718

Brameier, M., & Banzhaf, W. (2002). Explicit control of diversity and effective variation distance in linear genetic programming. In J. A. Foster, E. Lutton, J. Miller, C. Ryan, & A. Tettamanzi (Eds.), *Genetic programming* (pp. 37–49). Springer Berlin Heidelberg.

Brameier, M., & Banzhaf, W. (2007). *Linear genetic programming*. Springer-Verlag.

Branke, J., Deb, K., Dierolf, H., & Osswald, M. (2004). Finding knees in multi-objective optimization. In X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiňo, A. Kabán, & H.-P. Schwefel (Eds.), *Parallel problem solving from nature - ppsn viii* (pp. 722–731). Springer Berlin Heidelberg.

Castelli, M., Gonçalves, I., Manzoni, L., & Vanneschi, L. (2018). Pruning techniques for mixed ensembles of genetic programming models. In *Genetic programming* (pp. 52–67). Springer International Publishing. https://doi.org/10.1007/978-3-319-77553-1_4

Castelli, M., Manzoni, L., Silva, S., & Vanneschi, L. (2011). A quantitative study of learning and generalization in genetic programming. In S. Silva, J. A. Foster, M. Nicolau, P. Machado, & M. Giacobini (Eds.), *Genetic programming* (pp. 25–36). Springer Berlin Heidelberg.

Castelli, M., Vanneschi, L., & Silva, S. (2013). Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators. *Expert Systems with Applications*, *40*(17), 6856–6862. https://doi.org/https://doi.org/10.1016/j.eswa.2013.06.037

Cava, W. L., Orzechowski, P., Burlacu, B., de França, F. O., Virgolin, M., Jin, Y., Kommenda, M., & Moore, J. H. (2021). Contemporary symbolic regression methods and their relative performance.

Cavaretta, M., & Chellapilla, K. (1999). Data mining using genetic programming: The implications of parsimony on generalization error. *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, *2*, 1330–1337 Vol. 2. https://doi.org/10.1109/CEC.1999.782602

Celebi, M. E., & Aydin, K. (2016). Unsupervised learning algorithms. https://api.semanticscholar.org/CorpusID:64163481

Chen, Q., Xue, B., Shang, L., & Zhang, M. (2016). Improving generalisation of genetic programming for symbolic regression with structural risk minimisation. *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 709–716. https://doi.org/10.1145/2908812.2908842

Chen, Q., Xue, B., & Zhang, M. (2022). Rademacher complexity for enhancing the generalization of genetic programming for symbolic regression. *IEEE Transactions on Cybernetics*, *52*(4), 2382–2395. https://doi.org/10.1109/TCYB.2020.3004361

Chen, Q., Zhang, M., & Xue, B. (2017). Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. *IEEE Transactions on Evolutionary Computation*, *21*(5), 792–806. https://doi.org/10.1109/TEVC.2017.2683489

Chen, Q., Zhang, M., & Xue, B. (2019). Structural risk minimization-driven genetic programming for enhancing generalization in symbolic regression. *IEEE Transactions on Evolutionary Computation*, *23*(4), 703–717. https://doi.org/10.1109/TEVC.2018.2881392

Da Costa, L. E., & Landry, J.-A. (2006). Relaxed genetic programming. *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, 937–938. https://doi.org/10.1145/1143997.1144158

Daniel Rivero, C. F.-L., Enrique Fernandez-Blanco, & Pazos, A. (2020). Population subset selection for the use of a validation dataset for overfitting control in genetic programming. *Journal of Experimental & Theoretical Artificial Intelligence*, *32*(2), 243–271. https://doi.org/10.1080/0952813X.2019.1647562

Darwin, C. (1964). *On the origin of species: A facsimile of the first edition*. Harvard University Press. Retrieved May 21, 2024, from http://www.jstor.org/stable/j.ctvjf9xp5

Demir, F. (2005). A new way of prediction elastic modulus of normal and high strength concrete—fuzzy logic. *Cement and Concrete Research*, *35*(8), 1531–1538. https://doi.org/https://doi.org/10.1016/j.cemconres.2005.01.001

Dignum, S., & Poli, R. (2008). Operator equalisation and bloat free gp. In M. O'Neill, L. Vanneschi, S. Gustafson, A. I. Esparcia Alcázar, I. De Falco, A. Della Cioppa, & E. Tarantino (Eds.), *Genetic programming* (pp. 110–121). Springer Berlin Heidelberg.

Domingos, P. (2018). *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books, Inc.

Eiben, A. E., & Smith, J. E. (2015). Evolutionary computing: The origins. In *Introduction to evolutionary computing* (pp. 13–24). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-44874-8_2

Evett, M. P., Khoshgoftar, T., Chien, P.-d., & Allen, E. B. (1998). Gp-based software quality prediction. https://api.semanticscholar.org/CorpusID:14166245

Fitzgerald, J., Azad, R. M. A., & Ryan, C. (2013). Bootstrapping to reduce bloat and improve generalisation in genetic programming. *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, 141–142. https://doi.org/10.1145/2464576.2464647

Fogel, D. B. (1998). Evolutionary computation: The fossil record. https://api.semanticscholar.org/CorpusID:106864773

Foreman, N., & Evett, M. (2005). Preventing overfitting in gp with canary functions. *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, 1779–1780. https://doi.org/10.1145/1068009.1068307

Gagné, C., Schoenauer, M., Parizeau, M., & Tomassini, M. (2006). Genetic programming, validation sets, and parsimony pressure. In P. Collet, M. Tomassini, M. Ebner, S. Gustafson, & A. Ekárt (Eds.), *Genetic programming* (pp. 109–120). Springer Berlin Heidelberg.

Garey, M. R., & Johnson, D. S. (1990). *Computers and intractability; a guide to the theory of np-completeness*. W. H. Freeman & Co.

Gathercole, C., & Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In Y. Davidor, H.-P. Schwefel, & R. Männer (Eds.), *Parallel problem solving from nature — ppsn iii* (pp. 312–321). Springer Berlin Heidelberg.

Ghafourian, T., & Amin, Z. (2013). Qsar models for the prediction of plasma protein binding. *Bioimpacts*, *3*(1), 21–27. https://doi.org/10.5681/bi.2013.011

Goldberg, D. E. (1988). Genetic algorithms in search optimization and machine learning. https://api.semanticscholar.org/CorpusID:38613589

Gonçalves, I., & Silva, S. (2013). Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In K. Krawiec, A. Moraglio, T. Hu, A. Ş. Etaner-Uyar, & B. Hu (Eds.), *Genetic programming* (pp. 73–84). Springer Berlin Heidelberg.

Gonçalves, I., Silva, S., Melo, J. B., & Carreiras, J. M. B. (2012). Random sampling technique for overfitting control in genetic programming. In A. Moraglio, S. Silva, K. Krawiec, P. Machado, & C. Cotta (Eds.), *Genetic programming* (pp. 218–229). Springer Berlin Heidelberg.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* [Book in preparation for MIT Press]. MIT Press. http://www.deeplearningbook.org

Grimes, C. (1995). Application of genetic techniques to the planning of railway track maintenance work. *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, 467–472. https://doi.org/10.1049/cp:19951093

Han, J., Kamber, M., & Pei, J. (2011). *Data mining: Concepts and techniques* (3rd). Morgan Kaufmann Publishers Inc.

Harrison, D., & Rubinfeld, D. L. (1978). Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, *5*(1), 81–102. https://doi.org/https://doi.org/10.1016/0095-0696(78)90006-2

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction, second edition (springer series in statistics)*.

Haupt, S. (2009). Introduction to genetic algorithms. https://doi.org/10.1007/978-1-4020-9119-3_5

Hoerl, A. E., & Kennard, R. W. (2000). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *42*, 80–86. https://api.semanticscholar.org/CorpusID:28142999

Holland, J. H. (1992). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. https://api.semanticscholar.org/CorpusID:58781161

Iba, H., de Garis, H., & Sato, T. (1994). Genetic Programming Using a Minimum Description Length Principle. In *Advances in Genetic Programming, Volume 1*. The MIT Press. https://doi.org/10.7551/mitpress/1108.003.0017

Jones, T. (1995). Evolutionary algorithms, fitness landscapes and search.

Keijzer, M., & Babovic, V. (2000). Genetic programming, ensemble methods and the bias/variance tradeoff – introductory investigations. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, & T. C. Fogarty (Eds.), *Genetic programming* (pp. 76–90). Springer Berlin Heidelberg.

Kelly, M., Longjohn, R., & Nottingham, K. (2023). The uci machine learning repository. *URL https://archive. ics. uci. edu*.

Kooperberg, C. (1997). Statlib: An archive for statistical software, datasets, and information [Product/service evaluation]. *The American Statistician*, *51*, 98. https://link.gale.com/apps/doc/A19254911/AONE?u=anon~f45923b7&sid=googleScholar&xid=41701c55

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

Kronberger, G., Kommenda, M., & Affenzeller, M. (2011). Overfitting detection and adaptive covariant parsimony pressure for symbolic regression. In S. Gustafson & E. Vladislavleva (Eds.), *3rd symbolic regression and modeling workshop for gecco 2011* (pp. 631–638). ACM. https://doi.org/doi:10.1145/2001858.2002060

Lee, S.-C. (2003). Prediction of concrete strength using artificial neural networks. *Engineering Structures*, *25*(7), 849–857. https://doi.org/https://doi.org/10.1016/S0141-0296(03)00004-X

Li, J., Yanagisawa, K., Yoshikawa, Y., Ohue, M., & Akiyama, Y. (2021). Plasma protein binding prediction focusing on residue-level features and circularity of cyclic peptides by deep learning. *Bioinformatics*, *38*(4), 1110–1117. https://doi.org/10.1093/bioinformatics/btab726

Lourenço, J. M. (2021). *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. https://github.com/joaomlourenco/novathesis/raw/main/template.pdf

Lourenço, N., Ferrer, J., Pereira, F. B., & Costa, E. (2017). A comparative study of different grammar-based genetic programming approaches. In J. McDermott, M. Castelli, L. Sekanina, E. Haasdijk, & P. García-Sánchez (Eds.), *Genetic programming* (pp. 311–325). Springer International Publishing.

Luke, S., & Panait, L. (2002a). Fighting bloat with nonparametric parsimony pressure. In J. J. M. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, & J.-L. Fernández-Villacañas (Eds.), *Parallel problem solving from nature — ppsn vii* (pp. 411–421). Springer Berlin Heidelberg.

Luke, S., & Panait, L. (2002b). Lexicographic parsimony pressure. *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, 829–836.

Mahler, S., Robilliard, D., & Fonlupt, C. (2005). Tarpeian bloat control and generalization accuracy. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, & M. Tomassini (Eds.), *Genetic programming* (pp. 203–214). Springer Berlin Heidelberg.

McDonald, G. C., & Schwing, R. C. (1973). Instabilities of regression estimates relating air pollution to mortality. *Technometrics*, *15*(3), 463–481. Retrieved April 4, 2024, from http://www.jstor.org/stable/1266852

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst.*, *9*. https://api.semanticscholar.org/CorpusID:6491320

Miller, J. F., & Thomson, P. (2000). Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, & T. C. Fogarty (Eds.), *Genetic programming* (pp. 121–132). Springer Berlin Heidelberg.

Mitchell, T. (1997). *Machine learning*. McGraw-Hill. https://books.google.pt/books?id=EoYBngEACAAJ

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of machine learning*. The MIT Press.

Morvan, J.-M. (2008). *Generalized curvatures* (1st ed.). Springer Publishing Company, Incorporated.

Mousavi Astarabadi, S. S., & Ebadzadeh, M. M. (2015). Avoiding overfitting in symbolic regression using the first order derivative of gp trees. *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 1441–1442. https://doi.org/10.1145/2739482.2764662

Murphy, G., & Ryan, C. (2008). A simple powerful constraint for genetic programming. In M. O'Neill, L. Vanneschi, S. Gustafson, A. I. Esparcia Alcázar, I. De Falco, A. Della Cioppa, & E. Tarantino (Eds.), *Genetic programming* (pp. 146–157). Springer Berlin Heidelberg.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. The MIT Press.

Ni, J., & Rockett, P. (2015). Tikhonov regularization as a complexity measure in multiobjective genetic programming. *IEEE Transactions on Evolutionary Computation*, *19*(2), 157–166. https://doi.org/10.1109/TEVC.2014.2306994

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., & Moore, J. H. (2017). Pmlb: A large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, *10*(36), 1–13. https://doi.org/10.1186/s13040-017-0154-4

Paris, G., Robilliard, D., & Fonlupt, C. (2004). Exploring overfitting in genetic programming. In P. Liardet, P. Collet, C. Fonlupt, E. Lutton, & M. Schoenauer (Eds.), *Artificial evolution* (pp. 267–277). Springer Berlin Heidelberg.

Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, & E. Costa (Eds.), *Genetic programming* (pp. 204–217). Springer Berlin Heidelberg.

Poli, R., Langdon, W., & Mcphee, N. (2008). *A field guide to genetic programming*.

Raymond, C., Chen, Q., Xue, B., & Zhang, M. (2020). Adaptive weighted splines: A new representation to genetic programming for symbolic regression. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 1003–1011. https://doi.org/10.1145/3377930.3390244

Robilliard, D., & Fonlupt, C. (2002). Backwarding: An overfitting control for genetic programming in a remote sensing application. In P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, & M. Schoenauer (Eds.), *Artificial evolution* (pp. 245–254). Springer Berlin Heidelberg.

Rosca, J. P. (1996). Generality versus size in genetic programming. *Proceedings of the 1st Annual Conference on Genetic Programming*, 381–387.

Sambo, A. S., Azad, R. M. A., Kovalchuk, Y., Indramohan, V. P., & Shah, H. (2021). Evolving simple and accurate symbolic regression models via asynchronous parallel computing. *Appl. Soft Comput.*, *104*(100). https://doi.org/10.1016/j.asoc.2021.107198

Sharma, S., & Chahar, V. (2022). A comprehensive review on multi-objective optimization techniques: Past, present and future. *Archives of Computational Methods in Engineering*, *29*, 3. https://doi.org/10.1007/s11831-022-09778-9

Silva, S., Dignum, S., & Vanneschi, L. (2012). Operator equalisation for bloat free genetic programming and a survey of bloat control methods. *Genetic Programming and Evolvable Machines*, *13*(2), 197–238. https://doi.org/10.1007/s10710-011-9150-5

Silva, S., & Vanneschi, L. (2009). Operator equalisation, bloat and overfitting: A study on human oral bioavailability prediction. *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, 1115–1122. https://doi.org/10.1145/1569901.1570051

Silva, S., & Vanneschi, L. (2012). Bloat free genetic programming: Application to human oral bioavailability prediction. *Int. J. Data Min. Bioinformatics*, *6*(6), 585–601. https://doi.org/10.1504/IJDMB.2012.050266

Sotto, L., Kaufmann, P., Atkinson, T., Kalkreuth, R., & Porto Basgalupp, M. (2021). Graph representations in genetic programming. *Genetic Programming and Evolvable Machines*, *22*(4), 607–636. https://doi.org/10.1007/s10710-021-09413-9

Stitson, M. O., Gammerman, A., Vapnik, V., Vovk, V., Watkins, C., & Weston, J. (1999). Support vector regression with anova decomposition kernels. In *Advances in kernel methods: Support vector learning* (pp. 285–291). MIT Press.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second). The MIT Press. http://incompleteideas.net/book/the-book-2nd.html

Tibshirani, R. (2018). Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288. https://doi.org/10.1111/j.2517-6161.1996.tb02080.x

Tipping, M. E., & Bishop, C. M. (2002). Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *61*(3), 611–622. https://doi.org/10.1111/1467-9868.00196

Tohme, T., Liu, D., & YOUCEF-TOUMI, K. (2023). GSR: A generalized symbolic regression approach. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=lheUXtDNvP

Trujillo, L., Silva, S., Legrand, P., & Vanneschi, L. (2011). An empirical study of functional complexity as an indicator of overfitting in genetic programming. In S. Silva, J. A. Foster, M. Nicolau, P. Machado, & M. Giacobini (Eds.), *Genetic programming* (pp. 262–273). Springer Berlin Heidelberg.

Vanneschi, L., & Castelli, M. (2021). Soft target and functional complexity reduction: A hybrid regularization method for genetic programming. *Expert Systems with Applications*, *177*, 114929. https://doi.org/https://doi.org/10.1016/j.eswa.2021.114929

Vanneschi, L., Castelli, M., & Silva, S. (2010). Measuring bloat, overfitting and functional complexity in genetic programming. *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, 877–884. https://doi.org/10.1145/1830483.1830643

Vanneschi, L., & Gustafson, S. (2009). Using crossover based similarity measure to improve genetic programming generalization ability. *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, 1139–1146. https://doi.org/10.1145/1569901.1570054

Vanneschi, L., & Silva, S. (2023). *Lectures on intelligent systems*. Springer International Publishing.

Vapnik, V. (1999). An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, *10*(5), 988–999. https://doi.org/10.1109/72.788640

Vladislavleva, E. J., Smits, G. F., & den Hertog, D. (2009). Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, *13*(2), 333–349. https://doi.org/10.1109/TEVC.2008.926486

von Luxburg, U., & Schoelkopf, B. (2008). Statistical learning theory: Models, concepts, and results.

Whigham, P. (1999). Grammatically-based genetic programming.

Wishart, D. S., Knox, C., Guo, A. C., Shrivastava, S., Hassanali, M., Stothard, P., Chang, Z., & Woolsey, J. (2006). Drugbank: A comprehensive resource for in silico drug discovery and exploration. *Nucleic Acids Research*, *34*. https://doi.org/10.1093/nar/gkj067

Wolpert, D., & Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. https://doi.org/10.1109/4235.585893

Wu, Y., Lu, J., & Sun, Y. (2006). Genetic programming based on an adaptive regularization method. *2006 International Conference on Computational Intelligence and Security*, *1*, 324–327. https://doi.org/10.1109/ICCIAS.2006.294148

Yeh, I.-C. (1998). Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, *28*, 1797–1808. https://api.semanticscholar.org/CorpusID:135820588

Yoshida, F., & Topliss, J. (2000). Qsar model for drug human oral bioavailability1. *Journal of medicinal chemistry*, *43*, 2575–85. https://doi.org/10.1021/jm000564

Zaki, M. J., & Meira, J. W. (2020). *Data mining and machine learning* (2nd ed.). Cambridge University Press.

Zhang, B.-T., & Mühlenbein, H. (1995). Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, *3*, 17–38. https://doi.org/10.1162/evco.1995.3.1.17

Zhou, Z.-H., Wu, J.-x., Jiang, Y., & Chen, S.-F. (2001). Genetic algorithm based selective neural network ensemble.

Zou, H., & Hastie, T. (2005). Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *67*(2), 301–320. https://doi.org/10.1111/j.1467-9868.2005.00503.x

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

Universidade Nova de Lisboa