UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXVII CICLO DEL DOTTORATO DI RICERCA IN

APPLIED DATA SCIENCE AND ARTIFICIAL INTELLIGENCE

DIPARTIMENTO DI MATEMATICA, INFORMATICA, E GEOSCIENZE
FONDI DI ASSICURAZIONI GENERALI S.p.A.

Genetic Programming Across Domains: Leveraging Evolutionary Computation to Address Practical Problems

Settore scientifico-disciplinare: **ING-INF/05**

DOTTORANDO
LUIGI ROVITO

COORDINATORE
PROF. FRANCESCO PAULI

SUPERVISORE DI TESI
PROF. ANDREA DE LORENZO

ANNO ACCADEMICO 2023/2024

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Genetic Programming Across Domains: Leveraging Evolutionary Computation to Address Practical Problems

by

Luigi Rovito

Submitted for the degree of

Doctor of Philosophy

"Applied Data Science and Artificial Intelligence" PhD Program
XXXVII Cycle
Department of Mathematics, Informatics, and Geosciences
University of Trieste, Italy

Supervisor
Prof. Andrea De Lorenzo

October 2024

Abstract

Several modern technologies are based on Artificial Intelligence (AI) techniques, which are continuously studied and refined in many research fields. Among the subfields of AI, bio-inspired approaches, such as Genetic Programming (GP), have found great interest because of their effectiveness in discrete optimization problems that are prevalent in fields such as cryptography, code improvement, and Interpretable Machine Learning (IML). In this work, we investigate the application of GP methods in the three aforementioned topics.

Firstly, we employ GP to discover non-linear Boolean functions by exploring the search space of their Walsh transform-based representations for stream ciphers design. Boolean functions are essential components of secure cryptographic systems, and we specifically focus on optimizing non-linearity, a key security property, as well as balancedness, which provides resistance to statistical attacks. The proposed method demonstrates the capability of GP to evolve highly secure Boolean functions with a large number of variables, which are typically used in practical cryptographic scenarios.

Secondly, we try to improve the correctness of code generated by a Large Language Model (LLM) by leveraging a Genetic Improvement (GI) approach that, internally, employs a GP method defined as Grammatical Evolution (GE). This method starts from the code generated by an LLM, evaluates its correctness based on a set of user-provided test cases, and iteratively evolves improved solutions. Our results highlight the potential of combining GI and GE to enhance the reliability and correctness of automatically generated code, thus addressing one of the key limitations of current automatic code generation tools.

Thirdly, we propose a human-in-the-loop GP framework to evolve generic tree-based Machine Learning (ML) models that are evaluated in terms of both qualitative performance and interpretability. The interpretability of models is estimated by an Artificial Neural Network (ANN) that is trained with user feedback to capture the subjectivity of the user. This framework not only enables the discovery of models with competitive predictive performance but also provides users with interpretable solutions tailored to their specific background and requirements, which is a critical need in high-stakes applications.

Finally, we study how GP itself can be improved on its most famous problem, that is, Symbolic Regression (SR). Specifically, we extend a GP variant called Geometric Semantic Genetic Programming (GSGP) to develop Cellular Geometric Semantic Genetic Programming (cGSGP), which improves the diversity of the solutions evolved during the optimization process by imposing a neighborhood-based toroidal structure over the population. By limiting genetic operations to individuals in the same local neighborhood, cGSGP mitigates premature convergence and improves the exploration of the search space, leading to the discovery of more accurate symbolic models.

Our results highlight the effectiveness of GP in tackling the discussed problems, especially as regards building interpretable ML models, which is an urgent problem in high-stakes applications where the transparency of decisions is critical.

Dedication

For my family and friends, all of them.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Andrea De Lorenzo, whose guidance and support were invaluable throughout my PhD journey. I am also profoundly thankful to Prof. Luca Manzoni, Prof. Eric Medvet, and Dr. Marco Virgolin, whose expertise and collaboration as co-authors greatly contributed to my growth as a researcher.

A special thank you to my PhD student co-authors, Lorenzo Bonin, Giorgia Nadizar, Giovanni Pinna, and Damiano Ravalico, for their hard work and support in all the projects we undertook together during these three years. I am also grateful to the coordinator of the PhD program, Prof. Francesco Pauli, and to Prof. Luca Bortolussi, for giving me the opportunity to work at this esteemed university.

I would also like to extend my gratitude to Assicurazioni Generali S.p.A., who co-funded my scholarship and with whom I had the pleasure of collaborating. Lastly, I thank all my PhD student colleagues for the many shared experiences and enjoyable moments, and the professors and researchers in the PhD board for their continuous guidance and support.

Contents

List of Tables

List of Figures

Glossary

AI Artificial Intelligence.

ANF Algebraic Normal Form.

ANN Artificial Neural Network.

AST Abstract Syntax Tree.

BNF Backus-Naur Form.

CA Cellular Automata.

cGA Cellular Genetic Algorithm.

CGP Cartesian Genetic Programming.

cGSGP Cellular Geometric Semantic Genetic Programming.

DE Differential Evolution.

DL Deep Learning.

DNN Deep Neural Network.

EA Evolutionary Algorithm.

EC Evolutionary Computation.

EP Evolutionary Programming.

ES Evolution Strategies.

GA Genetic Algorithm.

GAN Generative Adversarial Network.

GE Grammatical Evolution.

GI Genetic Improvement.

GP Genetic Programming.

GPT Generative Pre-trained Transformers.

GSGP Geometric Semantic Genetic Programming.

GSO Geometric Semantic Operator.

HV Hyper-Volume.

IML Interpretable Machine Learning.

LFSR Linear Feedback Shift Register.

LGP Linear Genetic Programming.

LLM Large Language Model.

ML Machine Learning.

MLP Multi-Layer Perceptron.

ML-PIE Model Learning with Personalized Interpretability Estimation.

MSE Mean Squared Error.

NLP Natural Language Processing.

OTP One Time Pad.

PHI Proxy of Human Interpretability.

PRG Pseudo-Random Generator.

RL Reinforcement Learning.

RMSE Root Mean Squared Error.

RS Random Search.

SGD Stochastic Gradient Descent.

SR Symbolic Regression.

UML Unified Modeling Language.

VHDL Verilog Hardware Description Language.

WHT Walsh-Hadamard Transform.

XAI Explainable Artificial Intelligence.

List of Publications

[1] Giovanni Pinna, Damiano Ravalico, Luigi Rovito, Luca Manzoni, and Andrea De Lorenzo. "Enhancing Large Language Models-Based Code Generation by Leveraging Genetic Improvement". In: *Genetic Programming*. Ed. by Mario Giacobini, Bing Xue, and Luca Manzoni. Cham: Springer Nature Switzerland, 2024, pp. 108–124. ISBN: 978-3-031-56957-9. DOI: 10.1007/978-3-031-56957-9_7.

[2] Lorenzo Bonin, Luigi Rovito, Andrea De Lorenzo, and Luca Manzoni. "Cellular geometric semantic genetic programming". In: *Genetic Programming and Evolvable Machines* 25.1 (2024), pp. 1–32. DOI: 10.1007/s10710-024-09480-8.

[3] Giorgia Nadizar, Luigi Rovito, Andrea De Lorenzo, Eric Medvet, and Marco Virgolin. "An Analysis of the Ingredients for Learning Interpretable Symbolic Regression Models with Human-in-the-loop and Genetic Programming". In: *ACM Trans. Evol. Learn. Optim.* 4.1 (Feb. 2024). DOI: 10.1145/3643688.

[4] Luigi Rovito, Lorenzo Bonin, Luca Manzoni, and Andrea De Lorenzo. "An Evolutionary Computation Approach for Twitter Bot Detection". In: *Applied Sciences* 12.12 (2022), p. 5915. DOI: 10.3390/app12125915.

[5] Luigi Rovito, Andrea De Lorenzo, and Luca Manzoni. "Evolution of Walsh Transforms with Genetic Programming". In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. ACM, 2023, pp. 2386–2389. DOI: 10.1145/3583133.3596317.

[6] Luigi Rovito, Andrea De Lorenzo, and Luca Manzoni. "Discovering Non-Linear Boolean Functions by Evolving Walsh Transforms with Genetic Programming". In: *Algorithms* 16.11 (2023), p. 499. DOI: 10.3390/a16110499.

[7] Giorgia Nadizar, Luigi Rovito, Dennis G. Wilson, and Eric Medvet. "Interpretable Control Competition". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '24 Companion. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 11–12. ISBN: 9798400704956. DOI: 10.1145/3638530.3664051.

Chapter 1

Introduction

Contents

Many scientific research fields and modern technologies rely on methodologies and techniques that are referable to principles specific to the broad field of Artificial Intelligence (AI). As a matter of fact, many research fields are considered to be sub-fields of AI. Among them, bio-inspired approaches such as Evolutionary Algorithms (EAs) and, especially, Genetic Programming (GP), have found great interest in the scientific literature because of their effectiveness and applicability in solving complex, real-world problems that often require optimization or creative solutions. These approaches emulate principles of natural evolution, such as selection, mutation, and recombination, to iteratively improve candidate solutions to a problem.

One particularly compelling feature of GP lies in its adaptability across diverse fields of application. For example, GP has been effectively employed in discrete optimization problems, which are central to various domains of scientific and industrial relevance. Fields such as Boolean functions optimization for cryptography, code correctness improvement for enhancing software quality, interpretability optimization of Machine Learning (ML) models for high-stakes decision-making, and Symbolic Regression (SR) for deriving mathematical expressions from data have all benefited from GP-based methods. These applications often involve challenges that are computationally expensive or difficult to address through conventional techniques, underscoring the importance of advanced methodologies like GP.

In this work, we deeply investigate the application of GP methods in the four aforementioned topics, focusing on their unique challenges and providing solutions that extend the state-of-the-art in these domains. By doing so, this thesis aims to contribute both to the theoretical development of GP and to its practical application in addressing problems of high importance across cryptography, software engineering, and ML.

1.1 Research Problems

This thesis explores the application of GP, along with ML methodologies, to solve a given set of problems that are considered to be of high interest by various areas of the scientific community, as we are going to see in Chapter 3. These problems span multiple disciplines, ranging from cryptography to software engineering and Interpretable Machine Learning (IML), illustrating the versatility and broad applicability of GP in tackling diverse challenges. Below, we detail each research problem and its context:

1. **Non-Linear Boolean Functions Optimization**: in the context of symmetric cryptography, Boolean functions are critical components in the design of stream ciphers, which are widely used in secure communication systems. These functions play a fundamental role in ensuring the security properties of the cipher, such as resistance to linear and differential cryptanalysis. Non-linearity is a particularly important property, as higher non-linearity reduces the effectiveness of correlation attacks. Similarly, balancedness (ensuring that the truth table has the same number of 0s and 1s) strengthens resistance to statistical attacks. However, optimizing Boolean functions for both high non-linearity and balancedness becomes increasingly challenging as the number of variables of these Boolean functions grows, which is often the case in practical cryptographic systems. Addressing this problem is crucial to designing robust cryptographic primitives capable of withstanding modern attack strategies;
2. **Code Generation**: designing and implementing code is a complex activity that requires a human expert with a consolidated set of skills and experience. To this end, the problem of automatic code generation is considered extremely interesting since this task enables the developers to enhance their productivity and gain useful suggestions and insights during their coding sessions. Specifically, automatic code generation tools, such as those based on Large Language Models (LLMs), have gained popularity due to their ability to assist developers by generating code fragments, suggesting improvements, or even solving specific coding tasks. However, automatic code generation tools could provide incorrect or only partially correct code with logical errors or incomplete functionality, which needs to undergo an optimization or improvement to enhance its correctness. A problem in automatic code generation is the automatic enhancement of the correctness of code generated by an automatic code generation technique, e.g., an LLM. This is obtained by building new code starting from the code generated by the original tool. Improving the reliability and correctness of automatically generated code is particularly important in domains like safety-critical systems, where errors can have severe consequences;
3. **Interpretable Models Discovery**: ML algorithms are widely adopted in several real-world scenarios to perform predictions that help in taking informed decisions, from healthcare and finance to autonomous systems. While accuracy and performance metrics are often prioritized, the interpretability of models is becoming a critical requirement in high-stakes applications where understanding the reasoning behind predictions is essential for trust and accountability. For example, a healthcare provider may need to understand why an algorithm predicts a certain diagnosis before making a treatment decision. Therefore, directly interpretable models are considered extremely valuable in real-world cases, but their discovery and evaluation is difficult since there is no formal definition of interpretability, there are no objective metrics that are able to evaluate it, and, especially, interpretability highly depends upon the background and subjectivity of the user interacting with the model [1]. Consequently, the discovery of interpretable models tailored to the specific needs and backgrounds of their users is an urgent problem in IML. Tackling this requires innovative frameworks that balance predictive performance with subjective interpretability, making it possible to design models that are both effective and understandable;

4. **Symbolic Regression**: the problem of SR consists in the discovery of a mathematical formula that accurately interpolate a set of data points collected from a real-world environment. It has applications in fields such as physics, engineering, and economics, where interpretable models are preferred over black-box approaches. GP, with its ability to explore vast search spaces, is particularly well-suited for SR A variant of GP, called Geometric Semantic Genetic Programming (GSGP), is specifically tailored to solve SR tasks. It leverages semantic information during the evolutionary process to improve the accuracy of discovered formulae. However, one of the drawbacks of this method is the premature convergence to sub-optimal areas of the search space, probably due to a quick loss in diversity during the optimization. A problem in the context of GSGP for SR is the enhancement of this algorithm to mitigate the problem of the premature convergence with the goal of discovering more accurate formulae.

1.2 Contributions

For each of the aforementioned problem, we propose a method that is designed to provide a possible solution:

1. We employ GP to discover highly-secure Boolean functions by exploring the search space of their Walsh transform-based representations, and we show how these functions can be adopted as components of potentially confidentially-safe stream ciphers based on Pseudo-Random Generator (PRG);

2. Given a coding problem, we try to improve the correctness of the code generated by an LLM by leveraging a Genetic Improvement (GI) approach that, internally, employs a GP method defined as Grammatical Evolution (GE) to evolve and improve the solutions crafted by the LLM according to a set of user-provided test cases for the problem;

3. We propose a human-in-the-loop framework based on bi-objective GP to evolve generic tree-based ML models that are evaluated in terms of both qualitative performance (e.g., accuracy for classification models, error for regression models) and interpretability, where the latter is estimated by an Artificial Neural Network (ANN) that is trained, during the optimization process, with user feedback to steer the evolution towards models that are interpretable according to the subjectivity of the user;

4. We study how GP itself can be improved on its most popular problem, i.e., SR. Especially, we study a famous GP variant called GSGP that drives the evolution by accounting for the semantics of the individuals and we develop a method, namely, Cellular Geometric Semantic Genetic Programming (cGSGP), that improves the diversity of GSGP and mitigates the premature convergence of it by imposing a toroidal structure over the population of individuals, forcing the genetic operators to act only on individuals belonging to the same local neighborhood. We show that cGSGP meaningfully outperforms GSGP.

Our experimental analysis highlights that GP is an effective method in tackling the discussed problems, especially as regards building interpretable ML models, which is an urgent problem in high-stakes real-world applications that require models to

be as readable as possible [2, 3].

In Chapter 2 we describe the main concepts and principles regarding Evolutionary Computation (EC) and ML that are prevalent in our research topics, in Chapter 3 we perform a detailed literature review of our research problems, in Chapter 4 we describe our method for evolving non-linear Boolean functions, in Chapter 5 we describe our method for enhancing the correctness of code generated by LLMs, in Chapter 6 we describe our method for discovering interpretable ML models, in Chapter 7 we describe our method for mitigating the premature convergence of GSGP, and in Chapter 8 we write our conclusion.

Chapter 2

Conceptual Framework

Contents

Before delving into our literature review and proposed methods, we provide a brief overview of the main concepts needed to correctly understand and interpret the methodologies, techniques, and results that we are going to present in the next chapters.

2.1 Evolutionary Computation

Evolutionary Computation (EC) is a sub-field of Artificial Intelligence (AI) and computational intelligence that draws inspiration from the principles of natural selection and biological evolution to solve complex optimization and search problems. At its core, EC encompasses a family of algorithms that iteratively improve a population of candidate solutions by simulating the process of natural evolution, including mechanisms such as selection, mutation, recombination, and inheritance.

The fundamental components of an Evolutionary Algorithm (EA) include a population of individuals, each representing a potential solution to the problem at hand. These individuals are typically encoded as strings of symbols, such as binary strings, real-valued vectors, or more complex data structures, such as trees, depending on the specific application.

We begin by recalling the main components of a generic EA:

- **Individual**: a single solution $s \in S$ to the optimization problem, where S is the set of all possible solutions (i.e., the search space). The individual representation usually defines the specific type of EA that can be adopted;
- **Population**: a population $P \subseteq S$ of solutions (i.e., individuals) is evolved over time;
- **Fitness Function**: the fitness function $f : S \to \mathbb{R}^m$ evaluates how well each solution solves the problem at hand according to different criteria (when $m = 1$ the optimization problem is a single-objective one). This function is crucial as it guides the evolutionary process by determining which solutions are better and thus more likely to be selected for reproduction;

- **Selection**: selection is the process of choosing which individuals get to participate in the mating pool (i.e., the set of individuals that undergo the transformations provided by crossover and mutation) and create the next generation. Common selection methods include tournament selection, roulette wheel selection, and rank-based selection;
- **Crossover (Recombination)**: crossover involves combining parts of two parent solutions to create one or more offspring. This mimics the biological process of recombination, allowing for the mixing of good (or bad) traits from different parents;
- **Mutation**: mutation introduces random changes to individual solutions. This can help to maintain genetic diversity within the population and potentially discover new and better solutions;
- **Generation**: a generation refers to one complete cycle of evaluation, selection, crossover, and mutation. The process repeats for many generations, ideally leading to increasingly better solutions over time;
- **Termination Criterion**: the condition that, when true, ends the evolutionary process. This condition is typically checked at the end of each generation. Examples of termination criteria are number of generations, convergence of the population, time, and number of fitness evaluations (usually referred as computational budget).

An outline of a generic EA is detailed in Algorithm 1.

Algorithm 1 Generic EA (`EA`)

```
Initialize a population P of individuals
while termination criterion not met do
    Evaluate f on each individual in P
    Select individuals for the mating pool based on their fitness
    Apply crossover and mutation to create a new population
    Replace P with the new population
end while
return the best-so-far individual
```

2.1.1 Algorithms

There are several prominent types of EAs, each with its own distinct characteristics:

- **Genetic Algorithm (GA)** [4]: Genetic Algorithms are perhaps the most widely recognized form of EC. They typically operate on binary or real-valued representations and apply crossover and mutation to generate new candidate solutions;
- **Genetic Programming (GP)** [5]: Genetic Programming extends the concepts of GAs to the evolution of computer programs. Here, individuals are typically represented as tree structures. As we are going to see in Sec. 3.1, it has been successfully applied to a variety of tasks, including Symbolic Regression (SR), automated program synthesis, and classification;
- **Evolution Strategies (ES)** [6, 7, 8]: Evolution Strategies typically operate on vector representations and are characterized by the use of self-adaptive

mutation step sizes. Particularly, no crossover is performed and new candidate solutions are generated only via mutation;

- **Differential Evolution (DE)** [9, 10, 11]: Differential Evolution is another real-valued optimization algorithm that emphasizes the use of vector differences for generating new candidate solutions;
- **Evolutionary Programming (EP)** [12, 13, 14]: Evolutionary Programming is similar to GP but with a stronger emphasis on the evolution of finite state machines and similar structures. EP traditionally focuses on mutation as the primary operator, with less emphasis on crossover;
- **Grammatical Evolution (GE)** [15]: Grammatical Evolution is a form of GP that evolves programs in any language by mapping strings to syntax trees using a predefined grammar. This approach allows for the evolution of syntactically correct solutions and is particularly useful in areas where constraints or specific formats must be adhered to.

In our works, we use GP and GE to drive the evolutionary processes that compose our proposed approaches.

2.1.2 Genetic Programming

GP is an EA where a specific types of individuals, crossover, and mutation are defined. In GP, an individual represents a computer program, which is traditionally represented as a tree structure where internal nodes ($\mathcal{F}$) represent functions or operations, and terminal nodes or leaves ($\mathcal{T}$) represent inputs or constants (Fig. 2.1). This means that GP is particularly suitable for coding languages that naturally embody tree-based representations (e.g., Lisp [16]).

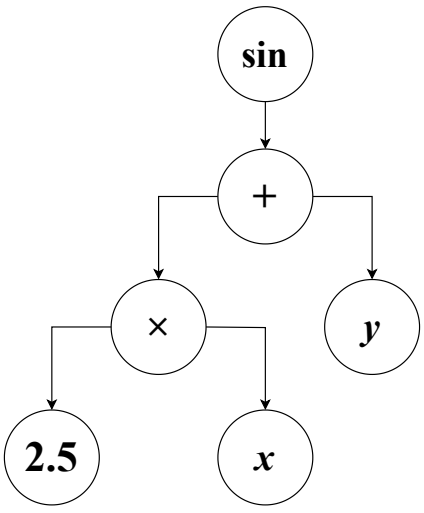

Figure 2.1: An example of GP tree representing the formula $\sin(2.5x + y)$.

Linear Genetic Programming (LGP) and Cartesian Genetic Programming (CGP) are popular variants of GP. In LGP, a program is a sequence of instructions from an imperative coding language that are typically executed in a linear fashion [17]. In CGP, a program is encoded by a graph that represents the program at hand

by using a two-dimensional grid of nodes that defines multiple inputs and multiple outputs for the program [18, 19].

A generic GP algorithm is identical to Algorithm 1 except for the fact that individuals are represented as trees and operations such as initialization, crossover, and mutation are tailored to work on this specific type of representation. Additional details on the specific initialization, selection, recombination, and mutation algorithms that are commonly employed with GP are described in Appendix B.

2.2 Machine Learning

Machine Learning (ML) is a branch of AI that focuses on the development of algorithms and statistical models that enable computers to perform tasks without explicit instructions. Instead of being programmed to execute a specific task, an ML system learns from data to make predictions or decisions. This ability to learn from experience and improve performance over time is what distinguishes ML from traditional rule-based programming.

ML serves various purposes depending on the specific goals of a task or application. Two of the primary purposes are predictive analysis and descriptive analysis, each addressing different aspects of data understanding and decision-making.

The primary goal of predictive analysis is to predict future outcomes based on historical data. By learning patterns and relationships in existing data, models can make informed guesses about what will happen in new or unseen situations. This type of analysis is essential in scenarios where accurate forecasts can drive better decision-making and strategy.

The main predictive tasks are categorized as:

- **Classification** [20]: the model predicts discrete labels or categories. For example, an email filtering system may predict whether an incoming email is spam or not. The system is trained on a labeled dataset where each email is marked as either "spam" or "not spam", and it uses this training to predict the label for new emails;
- **Regression** [21]: the model predicts continuous values. For example, a model could predict housing prices based on features such as location, size, and age of the house. In this case, the model outputs a numerical value that represents the estimated price;
- **Time Series Forecasting** [22]: predictive analysis is also used to forecast future data points in time series data, such as stock prices, weather conditions, or sales figures. These models learn trends and seasonal patterns from past data to predict future values.

Descriptive analysis focuses on understanding the underlying patterns, relationships, and structures within the data. Unlike predictive analysis, which looks forward, descriptive analysis is concerned with summarizing and interpreting historical data to provide insights and inform decisions. This type of analysis helps in identifying trends, correlations, and anomalies in the data.

The main descriptive tasks are categorized as:

- **Clustering** [23]: a technique used to group similar data points together based on certain features. For example, in customer segmentation, a company might use clustering to group customers with similar purchasing behaviors, allowing for targeted marketing strategies;
- **Dimensionality Reduction** [24]: descriptive analysis often involves reducing the number of variables or features in a dataset while preserving its essential characteristics. Techniques like principal component analysis help in simplifying the data, making it easier to visualize and interpret, especially when dealing with high-dimensional data;
- **Anomaly Detection** [25]: this involves identifying data points that deviate significantly from the norm. In Cybersecurity, for example, anomaly detection can be used to identify unusual network activity that might indicate a security breach;
- **Association Rule Learning** [26]: this is used to discover interesting relationships between variables in large datasets. A common example is market basket analysis, where a retailer might identify that customers who buy a certain product are likely to buy another related product.

More broadly, ML is typically divided into three main types:

- **Supervised Learning**: in supervised learning, the model is trained on a labeled dataset, where each input is paired with the correct output. The goal is to learn a mapping from inputs to outputs so that the model can predict the output for new inputs. Supervised learning is commonly adopted for predictive analysis;
- **Unsupervised Learning**: unsupervised learning involves training a model on a dataset without labeled outputs. The goal is to uncover hidden patterns or structures within the data. Clustering and dimensionality reduction are typical examples of unsupervised learning tasks. As such, unsupervised learning is commonly adopted for descriptive analysis;
- **Reinforcement Learning**: in reinforcement learning, an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties and learns to take actions that maximize cumulative rewards over time. This approach is commonly used in robotics, game playing, and autonomous systems.

In our works, we mainly adopt principles that adhere to supervised learning.

2.2.1 Supervised Learning

In supervised learning, the goal is to learn a function that maps input data to output labels based on a set of training examples. Let X denote the input space and Y denote the output space. A supervised learning problem consists of:

- A dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in X, \mathbf{y}_i \in Y, i \in [1..N]\}$ with $|D| = N$, where $\mathbf{x}_i$ represents the input features and $\mathbf{y}_i$ represents the corresponding labels. The dataset D is referred as training set and the elements that it contains are referred as training examples. Moreover, all the inputs $\mathbf{x}_i$ are independent and identically distributed (i.i.d.) samples drawn from an unknown probability

distribution $\mathcal{P}$ over X. Once a given $\mathbf{x}_i$ is sampled according to $\mathcal{P}$, the corresponding $\mathbf{y}_i$ is calculated by using an unknown labeling function $f : X \to Y$ such that $\mathbf{y}_i = f(\mathbf{x}_i)\ \forall i$;

- A hypothesis space H, which is a set of candidate functions $h : X \to Y$. Each function $h \in H$ represents a possible solution to the learning problem. Given a learning algorithm A, the application of A to D produces a hypothesis function $h_{A,D}$ that can be used to approximate f when new data are sampled according to $\mathcal{P}$;
- An error function $\mathcal{E} : Y \times Y \to \mathbb{R}_{\geq 0}$, which quantifies the difference between the predicted output $\tilde{\mathbf{y}}_i = h(\mathbf{x}_i)$ and the true output $\mathbf{y}_i$. The choice of the error function depends upon the nature of the problem (e.g., squared error for regression, cross-entropy for classification).

The objective of supervised learning is to find the hypothesis h^* within the hypothesis space H that minimizes the expected error over the distribution $\mathcal{P}$. Formally, this is defined as:

$$h^* = \arg\min_{h \in H} \mathbb{E}_{X \sim \mathcal{P}} \left[\mathcal{E}(h(X), f(X)) \right]$$

However, since both the distribution $\mathcal{P}$ and the labeling function f are unknown, we approximate the above objective by minimizing the empirical risk, which is the average error over the training dataset D. This leads to the empirical risk minimization problem:

$$h^* = \arg\min_{h \in H} \frac{1}{N} \sum_{i=1}^{N} \mathcal{E}(h(\mathbf{x}_i), \mathbf{y}_i)$$

Here, $\frac{1}{N} \sum_{i=1}^{N} \mathcal{E}(h(\mathbf{x}_i), \mathbf{y}_i)$ is known as the empirical risk or training loss. The ultimate goal is finding a hypothesis function that is able to generalize well on unseen data. However, the minimization of the aforementioned empirical risk over the entire hypothesis space H may lead to a h^* that over-fits D. Hence, practical scenarios involve minimizing the empirical risk over a restricted hypothesis space $\hat{H} \subset H$:

$$\hat{h}^* = \arg\min_{h \in \hat{H}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{E}(h(\mathbf{x}_i), \mathbf{y}_i)$$

Workflow

The workflow of supervised learning involves several steps. The first step is collecting and preparing the dataset:

- **Data Collection**: gathering data relevant to the problem at hand. The data should include both input features (independent variables) and their corresponding labels (dependent variables);
- **Data Splitting:** dividing the dataset into training, validation, and test sets. The training set is used to train the model, the validation set is used to tune

hyper-parameters, and the test set is used to evaluate the model performance on unseen data;

- **Data Cleaning**: handling missing values, correcting errors, and filtering out irrelevant data;
- **Feature Selection**: choosing the most relevant features from the dataset to reduce complexity and improve model performance;
- **Feature Extraction**: creating new features from the existing ones, often using domain knowledge;
- **Data Transformation**: converting data into a suitable format, which may include normalization, standardization, or encoding categorical variables;
- **Dimensionality Reduction**: applying techniques like principal component analysis to reduce the number of features while retaining essential information.

The choice of an appropriate model depends on the nature of the problem (e.g., classification or regression), the characteristics of the data, and the desired trade-offs between accuracy, interpretability, and computational efficiency. Common types of models include:

- **Linear Models**: such as linear regression for regression tasks and logistic regression for classification tasks;
- **Tree Models**: such as decision trees, which, together with linear models, are particularly suited when interpretability component depicts high importance;
- **Ensemble Models**: such as random forests or gradient boosting machines, which are flexible and powerful for both classification and regression because of the combination of several smaller models;
- **Support Vector Machines**: effective for high-dimensional spaces and cases where the decision boundary is non-linear;
- **Neural Networks**: suited for complex problems involving large amounts of data.

Once a model is selected, it is trained on the labeled training data. The way the training is performed depends upon the model and the specific learning algorithm adopted. Typically, we have the following phases:

- **Initialization**: initializing model variable components such as parameters and coefficients;
- **Optimization**: using an optimization algorithm to minimize a given loss function, which quantifies the difference between the predicted outputs and the true labels;
- **Hyper-parameter Tuning**: adjusting hyper-parameters to improve model performance, often by using techniques such as cross-validation.

After training, the model is evaluated to assess its performance:

- **Validation**: using the validation set to evaluate the model during training;
- **Testing**: assessing the final model performance on the test set to ensure that it generalizes well to new, unseen data. This usually involves calculating

metrics such as accuracy, precision, recall, F1-score (for classification tasks), or squared error (for regression tasks) to quantify model performance.

Once the model has been trained and validated, it can be deployed in a real-world environment to make predictions on new data:

- **Integration**: incorporating the model into an application or system where it can be used to make predictions;
- **Monitoring**: continuously tracking the model performance in production to detect any degradation in accuracy over time.

The final stage involves maintaining and improving the model over time:

- **Feedback Loops**: using feedback from users or the environment to improve the model;
- **Model Updates**: retraining or fine-tuning the model as more data becomes available or as the underlying data distribution changes;
- **Experimentation**: continuously experimenting with new models, features, or techniques to improve performance.

The supervised learning workflow is iterative, with ongoing refinement and retraining to adapt to new data and changing conditions.

To conclude, ML has revolutionized numerous fields by enabling systems that can perform complex tasks with high levels of accuracy and efficiency. Despite its success, ML also presents challenges, such as the need for large amounts of data, the risk of bias in models, and concerns over interpretability and transparency.

In the following chapter, we are going to review the scientific literature regarding our research problems, while in the next chapters, we are going to apply most of the principles, methodologies, and techniques that we have explored in this chapter to address the research problems outlined in Chapter 1. We, indeed, theorize that these principles and strategies may be effective in the resolution of these problems.

Chapter 3

Literature Review

Contents

In this chapter, we review the current scientific literature regarding the research topics introduced in the previous sections. We are going to analyze the main evolutionary-based approaches developed to address the problem of optimizing highly non-linear Boolean functions and the problem of generating and improving code, with the eventual support of a Large Language Model (LLM). Moreover, we are going to analyze how Evolutionary Computation (EC) can help in enhancing the interpretablity of Machine Learning (ML) models. Finally, we are going to analyze applications of approaches inspired by Cellular Automata (CA) to improve the effectiveness of evolutionary-based methods.

3.1 Genetic Programming

For each of the optimization problems that we are going to analyze, we propose a method based on an Evolutionary Algorithm (EA) defined as Genetic Programming (GP), which evolves computer programs represented by tree-like structures [5]. This meta-heuristic technique has found great interest in the scientific literature because of its flexibility and effectiveness in tackling a wide range of problems [27, 28, 29], including real-world ones [30, 31] as well as classification [32] and regression [33, 34] problems.

In this context, particular interest is depicted by the problem of Symbolic Regression (SR), which many researchers have tried to address by proposing several techniques that attempt to discover accurate mathematical expressions that can interpolate available data [35, 36], although the problem itself is considered difficult [37]. Especially, leveraging GP to solve SR problems has led to good results in several areas, such as biology [38], physics [39], robotics [40], wind turbines identification [41], finance [42], fluid dynamics inference [43], and climate modeling [44].

Provided that, GP is a promising approach in tackling optimization problems such the ones described in this work. As an additional point in favor, because of the way the models are built and the learning process is performed, in case the discovered solutions are contained in size, they may additionally exhibit interesting

interpretability properties [45, 46, 47, 48, 49, 50, 51], which, consequently, cover potential interest in high-stakes real-world applications, where, given their importance, the problem of ensuring that ML models decisions can be interpreted is urgent [2, 3].

3.1.1 Geometric Semantic Genetic Programming

Given the popularity and importance of SR, enhanced variants of vanilla GP were implemented to specifically address this problem in a better way. Among them, Geometric Semantic Genetic Programming (GSGP) [52, 53] showed that it can outperform GP in various tasks [54, 55, 56], gaining the interest of the scientific community [57]. GSGP explores the space of programs by using their semantic representations and by performing modifications that have direct effect on the semantic of the evolved programs.

GSGP was also combined with linear scaling [58, 59] and gradient descent [60, 61] to improve its performance, self-tuned to dynamically set the crossover and mutation rates [62], and modified to control bloating and reduce the exponential size of the individuals [63, 64, 65, 66, 67, 68], addressing one of the biggest issues in GSGP. Because of the availability of efficient implementations for the aforementioned techniques [69, 64], GSGP managed to gain better results than GP in various real-world problems [65, 56].

Despite its advantages over traditional GP, GSGP still faces certain limitations [70]. Specifically, while GSGP induces a unimodal fitness landscape, not all solutions within that landscape can be represented, resulting in the formation of local optima where evolution might become stagnant. Another drawback is that this variant can encourage early convergence towards certain individuals that dominate others, reducing the takeover time, which is the number of generations needed for one individual to dominate the population. As a result, the exploration of the fitness landscape may become restricted to mutation only, since the convex hull of the population, the only area accessible via crossover, may contract too much and drift away from the global optimum.

3.2 Boolean Functions Optimization

Let $\mathbb{B} = \{0, 1\}$, we begin by analyzing the problem of discovering a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ that exhibits high-quality security properties. This is of paramount importance in cryptography since most ciphers base their security properties on components that in turn are based on Boolean functions with specific characteristics, which enable the overall cipher architecture to be safe under certain circumstances [71, 72, 73].

A generic n-variables Boolean function can be represented by its truth table, that is, a binary string of size 2^n. In particular, 2^{2^n} is the number of all possible n-variables Boolean functions. Given that we are able to evaluate the goodness of a Boolean function w.r.t. a desired safety feature, a trivial approach for discovering the optimal Boolean function would be iterating over all possible Boolean functions of the same size and store the best one. This brute force approach is already unfeasible when $n > 5$. Furthermore, ciphers usually leverage Boolean functions with $n > 13$ [72, 73]. Hence, given the size of the search space, designing and building Boolean

functions having sound security characteristics requires more intelligent approaches than an exhaustive or a purely random one. Different strategies have been adopted to address this optimization problem.

Algebraic constructions, including the Rothaus construction [74] and the McFarland construction [75], were employed to develop various classes of Bent functions [76]. In their work, Li et al. [77] proposed a technique for generating n-variables Boolean functions with optimal algebraic immunity and established a lower bound on the number of such functions. Additionally, Chen et al. [78] introduced two classes of symmetric Boolean functions with optimal algebraic immunity, for which the algebraic degree and non-linearity were also determined. Tu et al. [79] put forth a combinatorial conjecture regarding binary strings, which, if proven true, would lead to the discovery of two classes of Boolean functions with optimal algebraic immunity: Bent functions and balanced functions, the latter of which possesses optimal algebraic degree, while Bent functions have the highest possible non-linearity since they achieve the covering radius bound.

Heuristic optimization techniques have been utilized to identify high-quality Boolean functions tailored to specific security requirements. Burnett et al. [80] enhanced the non-linearity of an initial Boolean function through incremental adjustments. Additionally, they developed a method to discover resilient Boolean functions by sequentially assembling valid Walsh spectra. Clark et al. [81] focused on optimizing functions that inherently possess strong cryptographic characteristics, with an effort to convert the most promising candidates into Boolean functions while retaining their original properties. In [82], Clark et al. employed a simulated annealing algorithm [83] to optimize high-quality Boolean functions. Lopez et al. [84] introduced a new diversity-aware meta-heuristic aimed at maximizing the non-linearity of Boolean functions, utilizing a cost function based on the Walsh-Hadamard Transform (WHT) to extract relevant information for the optimization process.

EC techniques offer promising alternatives to conventional optimization methods in the quest for Boolean functions with specific security attributes. Knevzevic et al. [85] conducted a survey detailing the application of EAs in both symmetric and asymmetric cryptography. Aguirre et al. [86] explored a multi-objective evolutionary strategy to identify balanced functions that meet various criteria, including non-linearity. Millan et al. utilized Genetic Algorithm (GA) to find balanced Boolean functions that satisfy both correlation immunity and the strict avalanche criterion [87], and they also used these algorithms to discover Boolean functions with high non-linearity [88]. Dimovski et al. [89] proposed a hill-climbing technique integrated with GA to search for highly non-linear Boolean functions. Asthana et al. [90] introduced a GA-based approach for constructing Boolean functions that fulfill the requirements of balancedness, correlation immunity, algebraic degree, and non-linearity. Behera et al. [91] applied GA in conjunction with a hybrid local search process to design Boolean functions with favorable auto-correlation and non-linearity characteristics. Mariot et al. [92] employed GA to evolve plateaued Boolean functions [93] by leveraging the spectral inversion technique described in [81], representing an individual chromosome as a permutation of a three-valued Walsh spectrum. Furthermore, Mariot et al. [94] implemented a particle swarm optimizer to generate Boolean functions with desirable cryptographic properties, particularly by updating particle positions while maintaining their Hamming weights to ensure that the generated functions remain balanced. This optimizer also uses hill-climbing to enhance

non-linearity and correlation immunity.

Miller et al. [95] conducted an empirical investigation into the use of Cartesian Genetic Programming (CGP) [18, 19], a variation of GP, for identifying general-purpose Boolean functions. Picek et al. [96] extended the work of Miller et al. [95] by examining the comparative effectiveness of CGP versus GP in learning highly non-linear Boolean functions. They also explored the performance of GA and GP with varying mutation operators and initialization strategies [97]. Hrbacek et al. [98] applied CGP to systematically generate Bent functions with up to 16 variables. Husa et al. [99] implemented a parallel version of Linear Genetic Programming (LGP) [17] to find Bent functions, focusing on maximizing the size of the functions, and conducted a comparative study of GP, LGP, and CGP for creating Boolean functions with an even number of inputs that exhibit high non-linearity, algebraic degree, balancedness, and correlation immunity [100].

Picek et al. [101] emphasized that among EAs, GP excels at discovering Boolean functions with non-linearity suitable for filter and combiner generators, as well as those with strong correlation immunity and minimal Hamming weight. Additionally, Picek et al. [102] explored the integration of algebraic and evolutionary approaches to search for secure Boolean functions. Mariot et al. [103] introduced several EAs to evolve Hyper-Bent functions, which offer greater security against approximation attacks, noting the difficulty of evolving these rare and complex functions. Additionally, Mariot et al. [104] used GA and GP to evolve Boolean functions that achieve perfect balancedness, focusing on evolving two types of solution encodings: truth table and weight-wise balanced representations. Carlet et al. [105] also explored evolving Boolean constructions by utilizing GP to generate balanced functions with high non-linearity. They [106] also investigated combining EAs with established algebraic constructions, specifically analyzing a recent generalization of the hidden weight Boolean function construction, demonstrating that evolutionary methods can significantly enhance the cryptographic properties of such functions.

In Chapter 4, we propose a novel method in which we represent Boolean functions by using their Walsh transform representations and we evolve these representations with GP to maximize the non-linearity while retaining the balancedness as best as possible. In addition, we present an in-depth experimental analysis in which several types of symbolic Walsh transform-based representations are examined, highlighting the most promising ones. We finally present an analysis of the composition of the truth tables that are implicitly built during the evolution to provide a justification on the reason why certain types of representations fail in reaching good results.

3.3 Code Generation

Code generation is, as the name suggests, the problem of automatically generating a correct code given a set of requirements such as a textual description of the problem or a set of test cases indicating valid input-output pairs for the output code.

Research in code generation has roots that extend back to the period preceding the development of the first electronic computers [107]. The topic began to attract considerable interest from the scientific community in the latter part of the twentieth century, particularly through studies in program synthesis [108, 109, 110, 111] and program transformation [112].

Since the problem of code generation (or program synthesis) is hard [113], a variety of tools for automatic code generation have been developed for different applications. These include generating optimized code from Event-B formal specifications [114], implementing design patterns [115], generating C code from mathematical models [116], utilizing Generative Adversarial Networks (GANs) to create code [117], producing Verilog Hardware Description Language (VHDL) code from Unified Modeling Language (UML) specifications [118] or high-level hardware descriptions [119], and generating code for web applications based on the model-view-controller architecture [120].

Automatic code generation has become a key area of interest within EC. Among the various approaches, GP is notably recognized as an influential EA specifically created to evolve programs. Building on this, Grammatical Evolution (GE) [15, 121] emerged and gained traction for its ability to represent and evolve programs based on a predefined grammar.

The work presented in [122] utilized an optimized hybrid EA to predict code execution time and enhance automatic code optimization for Ansor [123], an auto-scheduling system that builds on Apache TVM [124]. Additionally, a hybrid GA was employed in [125] to automate the generation of parallel code for handling regular control problems. CGP [18, 19] was used in [126] to evolve machine code for a simplified version of the MOVE processor. Moreover, multi-objective LGP was applied in [127] to generate assembly driver routines for devices in micro-controller-based systems. In [128], an EA was used to develop code for programmable logic controllers aimed at controlling various processes. As regards GE, it was applied to generate VHDL code for modeling digital circuits [129].

Promising tools that are gaining particular attention nowadays are the so-called Large Language Models (LLMs), which can be used for code generation tasks. They became popular with the advent of Transformers [130], a particular neural network architecture, and started a revolution within the Natural Language Processing (NLP) field. Since then, several research works and tools, including BERT [131], have been investigated and developed to address many NLP tasks [132]. LLMs were trained on large text corpora with the aim to improve their capabilities to predict missing text.

LLaMA [133, 134, 135], from Meta AI, and ChatGPT [136, 137], from OpenAI, are popular LLMs that have played pivotal roles in the advancement of automatic code generation. Although ChatGPT has demonstrated strong capabilities in this domain, it has also showed that an LLM has limitations [138]. Studies assessing the effectiveness of LLMs include an evaluation of Copilot utility in coding [139], an empirical investigation into the quality of code produced by ChatGPT [140], and an analysis of how the inherent non-determinism of ChatGPT-like tools impacts scientific reliability [141].

Assessing code generation presents considerable challenges, prompting the creation of various benchmarks. In [142], early LLMs, preceding LLaMA and ChatGPT, were tested using two benchmark datasets to expose their limitations. The PSB2 benchmark suite [143, 144], featuring 25 updated general program synthesis problems, is utilized to evaluate code generation techniques. For instance, the PSB2 suite has been employed to compare the performance of Copilot against evolutionary approaches like GP [145, 146]. The HUMANEVAL dataset [147] was designed

to assess functional correctness in code synthesis from doc-strings and has been used to test LLMs such as Codex. Notably, HUMANEVAL was employed in [148] to compare LLMs with GE. EvalPlus [149] extends HUMANEVAL by incorporating additional test cases to more thoroughly evaluate the functional correctness of code generated by LLMs. RepoMasterEval [150] is another benchmark designed to evaluate code completion techniques, derived from real-world TypeScript and Python repositories. EvoCodeBench [151] is a benchmark that mirrors real-world repositories in multiple aspects and mitigates data leakage. In [152], the authors developed a synthetic data generation algorithm called LintSeq that refactors existing code into a sequence of code edits and they fine-tuned small LLMs with these synthetic edit sequences. They showed that these fine-tuned models produce more diverse programs than baseline LLMs.

While many code generation tools, including those powered by LLMs, have demonstrated remarkable capabilities, their success in producing fully correct code often depends on the complexity of the problem and the quality of the input prompts. Genetic Improvement (GI) [153, 154], an approach derived from GP, is centered on optimizing existing code through evolutionary methods [155, 156]. GI has been applied to refine auto-generated code in languages like C++ [157, 158], C, Java [159], and Python [160]. Tools such as [161, 162] focus on bug fixing and optimizing time efficiency. GI was also utilized by Facebook in deploying the automated bug-fixing tool SapFix [163] and in efforts to reduce network usage [164]. Additionally, research in [165] examines the propagation effects of mutations in deeply nested code. Recent studies have explored the integration of GI techniques with LLMs to enhance and evolve code snippets generated by these models [166]. For example, [167] used modern LLMs to create new hybrid swarm intelligence optimization algorithms.

Recent studies have investigated the integration of LLMs with EAs [168], particularly focusing on the evolution and refinement of programs and code [169, 170, 171, 172, 173, 174]. Additionally, LLMs have been applied in EC for designing multi-objective evolutionary optimization operators [175], automatically generating optimization algorithms [176, 177], and tackling numerical optimization tasks [178]. Furthermore, LLMs have been explored for creating an Explainable Artificial Intelligence (XAI) dashboard in conjunction with GP [179], for tuning hyper-parameters in an EA [180], and for generating progressively complex test cases for curriculum learning aimed at training a GP agent [181]. In [182], the authors implemented a grammar-based evolutionary approach to explore the prompt space of LLMs, given the high impact the prompt quality has on LLMs outcomes.

Beyond evolving generated code, the refinement of prompts themselves is also crucial. Crafting an effective prompt can be challenging, but for an LLM, a well-crafted and detailed prompt can lead to significantly different results. EvoPrompting [183] was introduced to merge evolutionary prompt engineering with soft prompt-tuning, aiming to discover optimal neural architectures. EvoPrompt [184] serves as a framework for discrete prompt optimization, leveraging LLMs to evolve prompts. Similarly, Promptbreeder [185] was developed as a versatile, self-referential technique for evolving and adapting prompts tailored to specific domains.

In Chapter 5, we propose a technique in which we employ some of the most recent and performing LLMs, including both open-source and proprietary ones, and we leverage GI, with GE under the hood, to improve the code generated by these LLMs according to a set of user-provided test cases.

3.4 Interpretable Machine Learning

In the previous sections, we have analyzed applications of GP in two specific domains: Boolean functions optimization for cryptography purposes and code generation and improvement. Here, we inspect the literature regarding the research fields that address the problem of explainability in AI, which are XAI and, in our specific case, Interpretable Machine Learning (IML). Especially, since using ML in an irresponsible way may pose several risks [2], the field of XAI studies techniques to explain model predictions and generate models that are inherently interpretable [186, 187], where the latter strategy falls under the sub-field of IML.

This research field has gained paramount importance since high-stakes real-world applications usually rely on black-box models that can only be explained though specific explainability methods that, on the other hand, exhibit limitations and should be used with care [188, 189]. As such, interpretable models are preferable when qualitative performance meets the desired requirements [3].

The main challenge regarding building interpretable models is related to both the subjective and background of the user and the application involved [190, 1]. Plus, there could be a different trade-off between model qualitative performance and interpretability depending on the application and the user [191, 20]. Given the importance of this field and the urgency of the problem, several research works have tried to devise effective explainability techniques and interpretability-driven learning algorithms.

Innovative methods for explaining and interpreting ML model predictions have attracted significant attention in the scientific community [192, 193, 194, 195, 196]. The main strategies for achieving model explainability include either directly creating interpretable models through specialized learning algorithms or applying specific techniques (e.g., reverse engineering, feature attribution methods, counterfactual explanations) to black-box models to generate the necessary explanations [197, 187, 186, 198, 199]. While the latter approach is model-agnostic, it has the significant drawback of not providing a comprehensive understanding of how a model behaves for all possible inputs [3, 1]. In this discussion, we emphasize the former approach, focusing on the synthesis of inherently interpretable models.

Generally, decision trees, rule-based systems, and linear models are often seen as more suitable for creating interpretable models compared to more complex models like neural networks [200, 201, 202, 203, 204, 205]. However, even these relatively simpler models may still require additional justification, as their simplicity does not necessarily guarantee interpretability [1, 206]. Decision tree models can be made more interpretable by employing size reduction and pruning techniques [207, 208], or by optimizing a loss function that balances detection accuracy with model complexity [209, 210]. Similarly, linear models can be simplified by reducing the number of features they use [211, 212, 213, 214]. More broadly, many ML models, including linear models, regression trees, and decision trees, can be improved by eliminating irrelevant or redundant features. Neural network-based models can also be enhanced by removing unnecessary hidden layers and neurons.

In the realm of IML, EAs like GA and GP have attracted significant attention due to their ability to evolve models that are potentially interpretable (EC for XAI) [215, 216, 217, 45]. GP, in particular, has been extensively used to develop

inherently interpretable models such as decision trees [218] and classification rules derived from learning classifier systems [219]. One common approach to enhancing the interpretability of decision trees involves constraining the complexity of the tree structure. This is typically achieved by reducing the number of nodes in the tree, alongside other objectives, and by restricting the function set to include only simple functions [220, 221, 222, 223, 46, 224, 225, 226, 227].

Model components can also be assigned weights based on a predetermined scheme, where the weighted sum of these components can be used to estimate the interpretability of the model. Models that are less interpretable can be penalized during the evolutionary process [228, 229]. Additionally, simplification can be employed to streamline the learned models, thereby improving both interpretability and readability [230].

Recently, there has been significant research in applying interpretable EC to Reinforcement Learning (RL) scenarios [231, 232, 233, 234, 235, 236]. Custode et al. [237] integrated GE [15] with Q-learning [238] to evolve interpretable decision trees for RL tasks by decomposing the state-space. This method was further expanded by the authors to handle continuous action spaces [239]. Given that interpretable models often face challenges with raw data and high dimensionality, Custode et al. [240] introduced a framework featuring end-to-end pipelines of multiple interpretable models, co-optimized using EAs. This system enables the decomposition of the decision-making process in RL environments and the extraction of high-level, easily interpretable features from raw data. Additionally, ML techniques have been utilized to develop policies for pandemic containment [241, 242, 243]. In this context, Custode et al. [244] proposed a hybrid approach combining RL and EC methods to create interpretable policies specifically for managing the COVID-19 pandemic.

An alternative method involves a data-driven approach that demonstrates how to derive an interpretability formula based on human feedback, which can then be used to estimate the interpretability of mathematical expressions generated during a GP evolutionary process [51]. This approach was subsequently utilized by [237].

The framework introduced by [50] aimed to tackle the issue of interpretability being subjective to the individual observer. In their approach, an SR system is implemented where formulae are evaluated based on an interpretability estimator that is trained using user feedback. The system operates as a bi-objective GP process, with the first objective assessing the accuracy of the models and the second objective measuring interpretability through a neural network that is trained in parallel during the evolution. Formulae are paired and sampled using an active learning strategy that leverages estimator uncertainty, and these pairs are then presented to users [245, 246]. Users choose the most interpretable formula according to their personal understanding of interpretability. This feedback is utilized to train the interpretability estimator, which progressively learns to approximate a personalized Proxy of Human Interpretability (PHI).

Several studies have explored the development of AI systems that integrate human feedback during the training phase [247, 248, 249]. According to [225], including human input in the model synthesis process can facilitate the discovery of more interpretable models. Additionally, there are approaches where active learning is incorporated into GP [250, 251, 252], using the gathered labels to directly influence the objective function.

In Chapter 6, we build upon [50] and we show how we make the cited approach broader and more general. Especially, we design a technique that, differently from [50], can be applied to any type of GP problem and it is not limited to SR. This is possible since we implement novel ways of extracting numerical encodings from GP trees that are agnostic w.r.t. the specific problem and only account for the syntax of the tree. We demonstrate that using an estimator based on general, automatically-generated features allows for the approximation of any arbitrary PHI with a minimal number of training samples. While this method can be applied to a range of problem types, our experiments focused on SR due to its extensive significance and documented impact within GP literature.

3.5 Cellular Automata-inspired Approaches

In this last section, we investigate the literature review regarding approaches inspired by the discrete model of computation known as Cellular Automata (CA) [253, 254, 255], with particular focus on EC.

The concept of introducing spatial structure within the population of EAs has been previously investigated in the literature [256, 257, 258]. This approach involves limiting each individual interactions to a smaller subset of the population, referred to as its neighborhood. This is typically achieved by organizing the population into a grid structure, resulting in the creation of semi-isolated clusters of individuals that share similar characteristics.

In traditional GA [4], the concept of spatial structuring was introduced through the development of cellular GA (cGA) [256, 257, 258]. cGA has been applied to enhance optimization performance in well-known problems like the traveling salesman problem [259], where it was integrated with the simulated annealing algorithm [83]. Comprehensive studies on the effects of different neighborhood configurations and topologies on cGA performance across various problem domains were conducted in [256, 258]. Additionally, Alba et al. [256] introduced a dynamic version of cGA that adaptively modifies the exploration/exploitation balance during the evolutionary process.

Murata et al. [260] introduced a variant of GAs for multi-objective optimization that incorporates a cellular structure, known as C-MOGA, which builds upon the framework of the multi-objective GA. In C-MOGA, the selection operator is applied within the local neighborhood of each cell. Nebro et al. [261] later developed another method for multi-objective optimization, drawing inspiration from the traditional cGA, which they named MOCell. Mariot et al. [262] employed GA and GP to design orthogonal latin squares generated by CA.

Folino et al. [263] proposed a scalable parallel implementation of GP using a cellular structure, incorporating a load-balancing mechanism to evenly distribute computational tasks across processors. Their results demonstrated that this cellular approach could surpass the performance of both traditional GP and the island model [264], where separate evolutionary runs occur independently on four distinct population subsets. Additionally, studies by [265] and [266] showed that combining a spatial population structure with a local elitist replacement strategy can effectively mitigate the bloat phenomenon (i.e., the unnecessary growth in tree size without fitness improvement) in GP without sacrificing performance.

Beyond GA and GP, a range of other EAs have utilized the cellular model to manage the spread of solutions within the population [267, 268, 269], highlighting that the concept of a cellular structure can be broadly applied to populations of solutions, regardless of the specific algorithm being used.

Diversity within a population can also be enhanced through a process called speciation. This idea is inspired by the natural world, where ecosystems are made up of unique physical spaces, or niches, each with its own characteristics. These diverse niches allow for the development of different species, with members of each species exhibiting shared traits. Within these niches, resources are limited, driving competition among individuals.

Similarly, in an EA, the population can be intentionally divided into distinct niches made up of individuals with comparable structural traits. This setup restricts crossovers to occur only between individuals of the same species.

Della Cioppa et al. [270] introduced an adaptive species discovery strategy aimed at overcoming some of the limitations of traditional niching methods that depend on prior knowledge of the fitness landscape. Inspired by the design principles of the NEAT algorithm [271], authors in [272] employed speciation to control program growth in GP. Their algorithm, neat-GP, ensures that complexity develops only when necessary by gradually dividing the population into species based on similarities in size and structure. Juárez-Smith et al. [273] enhanced neat-GP by integrating a local search operator to improve solution quality. In [274], the authors proposed a network distance metric to speciate a population of artificial gene regulatory networks, demonstrating that speciation promotes diversity and maintains smaller individuals. Martins et al. [275] combined GA with speciation and a grid pattern recognition method to reduce investment risks and increase profits, and they also analyzed the differences between speciated and non-speciated approaches. Additionally, Wickman et al. [276] utilized speciation to evolve a diverse set of policies within RL. Pietropolli et al. [277] also proposed to use, instead of a flat toroidal grid, substrates where some empty cells (i.e., barriers) never host individuals to slow down the propagation of genetic traits and improve diversity.

Although numerous studies have explored cellular EAs, determining whether the cellular structure in EAs represents the state-of-the-art is not straightforward. Various research efforts have investigated the impact of selection pressure and sampling strategies in both traditional EAs [278, 279] and cellular-based EAs [280, 281].

The cellular framework in EAs can notably affect the takeover time, which refers to the number of generations needed for a single dominant individual to completely spread its copies throughout the population. A shorter takeover time indicates a rapid decline in population diversity. In cellular-based EAs, the spatial arrangement plays a crucial role in regulating both takeover time and selection pressure, as demonstrated in studies such as [281].

In Chapter 7, building on the fact that premature convergence to local areas is one of the major limitations of GSGP [70], we present a method based on imposing a cellular structure to the population of GSGP, since, as we have previously noted, this kind of strategy may help to increase the takeover time and avoid a quick diversity decade. To the best of our knowledge, this is the first attempt in applying a cellular structure to GSGP.

Chapter 4

Non-Linear Boolean Functions Optimization via Walsh Transforms Evolution

Contents

The first area of exploration involves the use of Genetic Programming (GP) for discovering non-linear Boolean functions [282, 283], a critical task in many areas of cryptography and digital circuits design.

4.1 Background

In symmetric cryptography, the communication, which is supposed to happen within an unsafe channel, is protected with encryption and decryption by using a shared secret key that only the participants involved in the communication must know [71]. A common problem is designing a symmetric cipher, in our case a stream cipher, that is confidentially-safe against eavesdropping attacks. This means that the stream cipher, which elaborates each message as a single entire block one bit at a time, must guarantee that even if an attacker manages to take an encrypted message, without knowing the secret key, it would be as hard as possible to retrieve information concerning the plain-text.

Boolean functions are important building blocks of several stream cipher-based architectures and the quality of their security properties enable the stream ciphers to be safe in practical scenarios. Among these properties, non-linearity and balancedness depict particular importance [71, 72, 73].

We focus our study on the one-time key Vernam stream cipher [284] combined with a specific Pseudo-Random Generator (PRG) [285, 286] defined as Combiner model [72], which internally employs a series of Linear Feedback Shift Registers (LFSRs) [287]. This stream cipher architecture is similar to the One Time Pad (OTP), which satisfies Shannon's perfect secrecy property [288], but the shared

secret key does not need to match the length of the plain-text message since the PRG has the task to expand the secret key so that the standard version of OTP can be executed to encrypt and decrypt the message, making this architecture usable in practical cases. A Vernam stream cipher is considered to be confidentially safe against eavesdroppers if and only if the employed PRG is safe, that is, unpredictable. Additional details on the Vernam stream cipher and the security definitions that can be applied with it are described in Appendix A.

4.1.1 Combiner Model

A LFSR is an unsafe PRG [289]. It produces a single bit at every clock cycle starting from a given binary string called "seed". For example, given a state represented by the current seed stored in the LFSR, at every clock cycle, the left-most bit is returned as output. Then, the registry executes a left shift operation, and the right-most bit is populated with bit-wise XOR operations performed on a subset of the seed. This way, if the registry works for d clock cycles, it will generate a d-bit long binary string.

Even though the LFSR is unsafe, by combining n LFSRs with an n-variables Boolean function it is possible to build a potentially safe PRG, that is, the Combiner model. The seed is an s-sized binary string (i.e., the secret key). The LFSRs are initialized with this seed and, at each clock cycle, the LFSRs output one bit for each in parallel. Then, the Boolean function f produces a single bit starting from the outputs of the LFSRs. This process is iterated for d clock cycles with $d >> s$ in order to produce an expanded version of the initial key that can be employed to encrypt a d-sized binary string representing a given plain-text message. The Combiner model is an unpredictable, or safe, PRG if the Boolean function f exhibits highly secure cryptography properties. Therefore, discovering a Boolean function characterized by high-quality safety features depicts high interest in cryptography since it can be adopted as component of the Combiner model, which, in conjunction with the PRG-based Vernam cipher, enables the construction of a stream cipher that is confidentially safe against eavesdroppers.

4.1.2 Boolean Functions

Let $\mathbb{B} = \{0,\ 1\}$ be the set of binary values, where zero stands for *False* and one stands for *True*. Let $\oplus$ be the sum operator (logical XOR) defined over $\mathbb{B}$, and $\wedge$ the product operator (logical AND). Let $f : \mathbb{B}^n \to \mathbb{B}$ be a generic n-variables Boolean function that can be represented by using its truth table, that is, a vector belonging to $\mathbb{B}^{2^n}$.

For the sake of simplicity, in this chapter, we are going to use n to denote the number of variables of a Boolean function, and f to denote a generic Boolean function. Moreover, we are going to use the terms "binary vector" and "binary string" interchangeably.

A Boolean function f with n variables can be formulated by: a $\mathbf{\Omega}_f \in \mathbb{B}^{2^n}$ vector holding the truth table of f, a multivariate polynomial defined as Algebraic Normal Form (ANF) of f (defined as a sum of products over $\mathbb{B}$), or a Walsh transform [81]. Let $\mathbf{w} \cdot \mathbf{x} = \bigoplus_{i=1}^{n} \mathbf{w}_i \wedge \mathbf{x}_i$ be the dot product defined over $\mathbb{B}^n$, a Walsh Transform is defined as $\hat{F} : \mathbb{B}^n \to \mathbb{R}$:

$$\hat{F}(\mathbf{w}) = \sum_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x})(-1)^{\mathbf{w}\cdot\mathbf{x}} \tag{4.1}$$

By applying $\hat{F}$ to all the $\mathbf{w} \in \mathbb{B}^n$, we produce the Walsh spectrum of f, i.e., the vector of Walsh coefficients $\mathbf{S}_f \in \mathbb{R}^{2^n}$. Given a generic $\mathbf{S}_f$, it is possible to build $\hat{\mathbf{\Omega}}_f \in \mathbb{Z}^{2^n}$, which is the polar form version of the truth table of f. The polar form is a truth table consisting only of -1 (*True*) and 1 (*False*). We can retrieve the polar form with the application of the Inverse Walsh Transform $\hat{F}^{-1} : \mathbb{B}^n \to \mathbb{R}$:

$$\hat{F}^{-1}(\mathbf{x}) = 2^{-n} \sum_{\mathbf{w}\in\mathbb{B}^n} \hat{F}(\mathbf{w})(-1)^{\mathbf{w}\cdot\mathbf{x}} \tag{4.2}$$

There are no guarantees that calling Eq. (4.2) by providing a generic spectrum $\mathbf{S}_f$ as input leads to a truth table in polar form. Generally, Eq. (4.2) outputs a polar form-based pseudo-Boolean function $\tilde{f} : \mathbb{B}^n \to \mathbb{R}$.

Given a pseudo-Boolean function $\tilde{f}$, several nearest Boolean functions can be built starting from it. The standard strategy consists in defining a single f as follows:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \tilde{f}(\mathbf{x}) > 0 \\ -1 & \text{if } \tilde{f}(\mathbf{x}) < 0 \\ -1 \text{ or } 1 \text{ at random} & \text{if } \tilde{f}(\mathbf{x}) = 0 \end{cases} \tag{4.3}$$

When $\tilde{f}(\mathbf{x}) = 0$ we say that there is uncertainty by $\tilde{f}$ as regards the computation of the truth value of $\mathbf{x}$. In Eq. (4.3), the uncertain positions are filled by adopting a random criterion (i.e., unbiased uncertainty filling). However, uncertain positions could also be filled by leveraging smarter strategies. After the nearest Boolean function f has been built, the corresponding Walsh Spectrum can be obtained by applying Eq. (4.1).

Let $w_H : \mathbb{B}^n \to \mathbb{N}$ be the Hamming weight of a given binary vector, that is, the number of 1s contained in it. The Hamming weight of f can be computed by applying the Hamming weight to $\mathbf{\Omega}_f$, i.e., the truth table of f.

A Boolean function f is balanced if and only if $\hat{F}(0) = 0$, i.e.,

$$|\{b \in \mathbf{\Omega}_f \mid b = 0\}| = |\{b \in \mathbf{\Omega}_f \mid b = 1\}|$$

Under this condition, $w_H(f) = 2^{n-1}$.

We define the unbalancedness degree of a Boolean function as

$$\big|\ |\{b \in \mathbf{\Omega}_f \mid b = 0\}| - |\{b \in \mathbf{\Omega}_f \mid b = 1\}|\ \big|$$

The truth table is perfectly balanced when the unbalancedness degree is zero.

The Hamming-based distance between two binary vectors of equal size is the number of positions in which the corresponding bits are not equal. The non-linearity of f can be formalized as the Hamming-based distance between f and linear functions [72]:

$$\bar{\ell}(f) = 2^{n-1} - \frac{1}{2} \max_{\mathbf{w} \in \mathbb{B}^n} |\hat{F}(\mathbf{w})| \tag{4.4}$$

The non-linearity $\bar{\ell}$, when n is even, is bounded by the Covering Radius bound (U_n^{crb}) [290], which states that $\bar{\ell} \leq 2^{n-1} - 2^{\frac{n}{2}-1}$. When n is odd, the maximum non-linearity of the n-variables Boolean functions is lower bounded by $2^{n-1} - 2^{\frac{n-1}{2}}$ ($L_n^{\bar{\ell}}$) and upper bounded by $2\lfloor 2^{n-2} - 2^{\frac{n}{2}-2} \rfloor$ ($U_n^{\bar{\ell}}$) [291].

4.2 Problem Formulation

The number of all possible n-variables Boolean functions is 2^{2^n}. Among them, we need to discover the Boolean function with the best non-linearity. However, an exhaustive search is already unfeasible when $n > 5$, and in practical scenarios Boolean functions have $n > 13$ [72, 73], meaning that there is the need of an effective exploration strategy. Moreover, the optimal non-linearity, in the cases in which this value is theoretically known, still has not been reached yet [73].

An Evolutionary Algorithm (EA) that adopts a proper Boolean function representation can help in driving the search of non-linear Boolean functions. By performing different combinations of EAs and Boolean representations we can explore the search space by using different strategies.

We aim to discover Boolean functions with high non-linearity and balancedness. We are interested in balanced Boolean functions since unbalanced functions are characterized by statistical bias that can be exploited by attackers. Furthermore, we are interested in highly non-linear Boolean functions since they can more easily defend themselves from fast-correlation attacks [73].

4.3 Proposed Method

We employ Genetic Programming (GP) [5] to evolve symbolic formulae. In particular, these trees or formulae represent Walsh transforms of general pseudo-Boolean functions, which can then be used to identify specific Boolean functions. Hence, a given individual or solution is a symbolic formula representing a Walsh transform. Moving forward, we will use the terms "tree" and "formula" interchangeably.

As regards the GP structure, we explore two function sets:

$$\mathsf{F1} = \{+,\ \times,\ (.)^2\}$$
$$\mathsf{F2} = \{+,\ -,\ \times,\ \%^*,\ \texttt{even},\ \texttt{odd}\}$$

where

$$\%^*(a, b) = \texttt{sign}(b)(a\ \%\ (|b| \text{ if } b \neq 0 \text{ else } 1))$$

$$\texttt{even}(x) = \begin{cases} 1, & \text{if } x \text{ is even} \\ 0, & \text{otherwise} \end{cases}$$

$$\texttt{odd}(x) = \begin{cases} 1, & \text{if } x \text{ is odd} \\ 0, & \text{otherwise} \end{cases}$$

The Walsh transform gets an n-sized binary string as input. This input can be encoded in the following ways, resulting in different terminal sets:

1. A terminal set of one variable representing an integer between 0 and $2^n - 1$, i.e., the integer encoded by a given binary string of size n (N);
2. A terminal set of n variables representing the binary positions of the input string (B);
3. A terminal set of n variables representing the binary positions of the input string in polar form (P).

For each terminal set, we create an expanded version of it with ephemeral random constants uniformly sampled between -1 and 1 (ERC). A tree syntactical structure is the composition of a function set and a terminal set, denoted with the notation $\mathcal{S}^{<\text{terminal\_set}>}_{<\text{function\_set}>}$.

In our method, a tree represents a given $\hat{F}$. A tree can be applied to each $\mathbf{w} \in \mathbb{B}^n$, resulting in the retrieval of a Walsh spectrum $\hat{S}_f \in \mathbb{Z}^{2^n}$. This spectrum may initially contain real values, which are subsequently rounded to the nearest integers. By employing the spectral inversion technique, Eq. (4.2) can be utilized on $\hat{S}_f$ to derive, in general, a pseudo-Boolean function $\hat{f}$. To compute the nearest Boolean function from $\hat{f}$, various strategies can be used. These strategies vary based on the criterion employed to assign values to the entries of $\mathbf{\Omega}_f$ where $\hat{f}(\mathbf{w}) = 0$:

1. **Unbiased uncertainty filling**: the nearest Boolean function is calculated by leveraging Eq. (4.3). This strategy was already explored in [81];
2. **Biased (toward balancedness) uncertainty filling**: we propose this strategy to improve or, eventually, preserve the balancedness when computing nearest Boolean functions. The idea is that uncertain positions can be used to balance $\mathbf{\Omega}_f$. Let $\hat{\mathbf{\Omega}}_f$ be the polar form-based truth table of $\hat{f}$. Let:

$$\begin{aligned} v_1 &= |\{b \in \hat{\mathbf{\Omega}}_f \mid b < 0\}| \\ v_0 &= |\{b \in \hat{\mathbf{\Omega}}_f \mid b > 0\}| \\ u &= |\{b \in \hat{\mathbf{\Omega}}_f \mid b = 0\}| \end{aligned}$$

Our goal consists in discovering $\mathbf{u}^* \in \{-1,\ 1\}^u$, i.e.:

$$\mathbf{u}^* = \arg \min_{\mathbf{u} \in \{-1,\ 1\}^u} |(v_1 + u_1) - (v_0 + u_0)|$$

where

$$\begin{aligned} u_1 &= |\{b \in \mathbf{u} \mid b = -1\}| \\ u_0 &= |\{b \in \mathbf{u} \mid b = 1\}| \end{aligned}$$

The unbalancedness degree is not affected by the way -1 and 1 are sorted in $\mathbf{u}^*$. Building on that, a given $\mathbf{u}^*$ can be computed deterministically (e.g.,

$\mathbf{u}^* = [-1^{u_1}, 1^{u_0}]$) and, to enhance diversity, a random permutation of it can be performed, resulting in $\mathbf{u}_p^* \in \{-1,\ 1\}^u$. We use $\mathbf{u}_p^*$ to assign the uncertain entries of $\mathbf{\Omega}_f$:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \tilde{f}(\mathbf{x}) > 0 \\ -1 & \text{if } \tilde{f}(\mathbf{x}) < 0 \\ \mathbf{u}_p^*[\texttt{ind}(\mathbf{x})] & \text{if } \tilde{f}(\mathbf{x}) = 0 \end{cases} \tag{4.5}$$

where $\mathbf{u}_p^*[\texttt{ind}(\mathbf{x})]$ is the value in $\mathbf{u}_p^*$ located at the position given by $\texttt{ind}(\mathbf{x})$, where the latter represents an integer between 0 and $u-1$. `ind` is defined only for uncertain binary strings w.r.t. $\hat{f}$ and provides the ranking in which these strings appear in $\hat{\mathbf{\Omega}}_f$.

Once the truth table $\mathbf{\Omega}_f$ of the nearest Boolean function of $\hat{f}$ is calculated, Eq. (4.1) is applied to retrieve the associated Walsh spectrum S_f. The fitness function is the non-linearity $\overline{\ell}$ Eq. (4.4) evaluated by using S_f and the optimization is configured to maximize the fitness function.

4.4 Experimental Analysis

In our experimental phase, we set Random Search (RS) as baseline and we compare it with GP. We test both unbiased and biased uncertainty filling strategies. We test n ranging from 6 to 16, which are the number of bits that are usually employed in real cases. For each combination of n, tree structure, and uncertainty filling strategy, we perform 30 repetitions of both RS and GP.

RS generates 2500 random formulae and selects the best one according to the non-linearity. GP runs for 5 generations with a population size of 500. We set the computational budget so that RS and GP have the same one for fair comparison. We evolve the GP for a few generation to test the approach with a low budget, enabling running this method even with commodity hardware. We use tournament selection with tournament size 5, sub-tree crossover (executed with probability 0.80), and sub-tree mutation (executed with probability 0.30). The maximum depth is always 5 to limit the size of the trees (the depth of the root is 0). We use these parameters since they are standard in GP evolution.

The experimental phase is articulated in three areas:

1. We analyze the distribution of the best overall solutions and we analyze whether GP is meaningfully better than RS;
2. We analyze the unbalancedness degree of the best overall solutions and we compare the two filling strategies;
3. We delve into the mean percentage of uncertain positions in the discovered pseudo-Boolean functions to provide a justification of part of the obtained results.

Python 3.9 and PyMOO [292] are used for the implementation, which is available at https://github.com/lurovi/BooleanCryptoGP.

4.4.1 Non-linearity Distribution

The goal of this analysis is to compare RS and GP to figure out the cases in which the evolution based on GP is able to meaningfully perform better than RS while discovering higher quality solutions despite the low computational budget employed.

We compare the distributions of non-linearity of the best individuals identified with RS and GP. We adopt the biased uncertainty filling strategy. For each n, we show a box-plot with the p-values of a Wilcoxon signed-rank test [293] performed on the non-linearity values across all the repetitions. In this case, the statistical test assesses whether GP is better than RS in a statistically significant way. To make the plot readable, a scaling between 0 and $U_n^{\bar{\ell}}$ is performed on each non-linearity value.

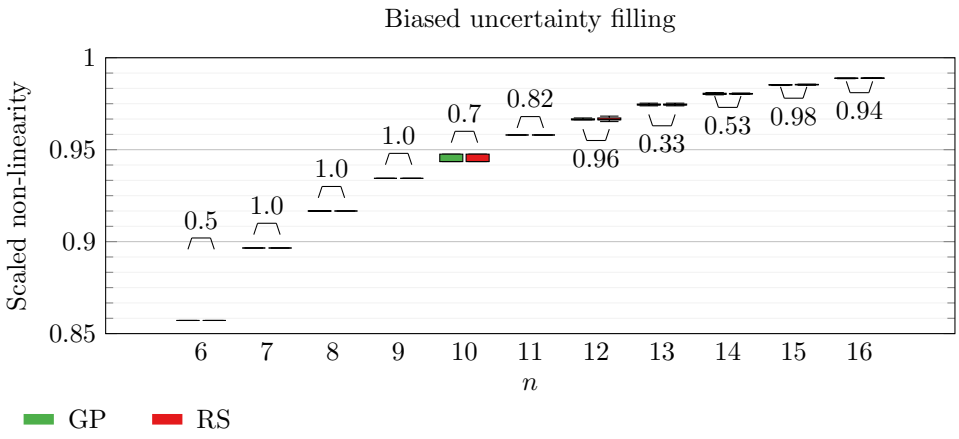

Figure 4.1: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F1}}^{\mathsf{B}}$.

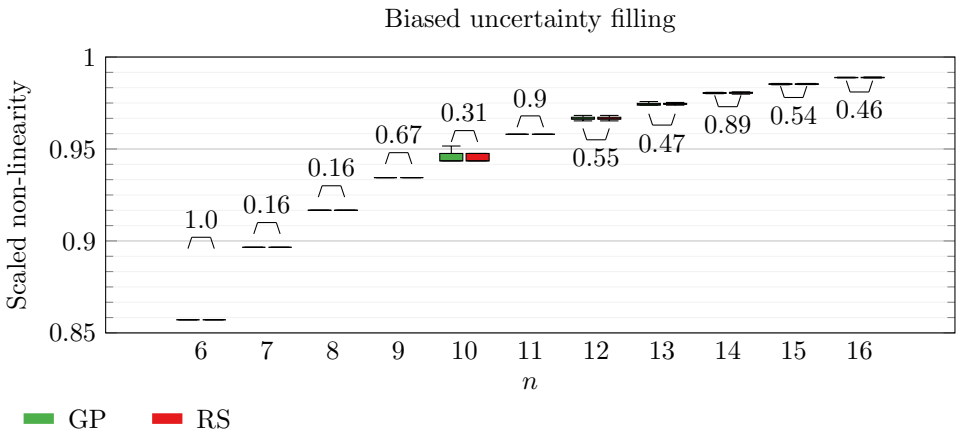

Figure 4.2: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F1}}^{\mathsf{B,ERC}}$.

We can see that the only Walsh transform representations that can take advantage of GP are $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N,ERC}}$ and $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N}}$ (Fig. 4.4, Fig. 4.3). The other symbolic representations show that GP is not meaningfully better than RS. Therefore, $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N,ERC}}$ and $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N}}$ provide the best results in this scenario, and thus future research efforts should probably focus on these types of configurations. Interestingly, non-linearity tends to improve as n increases in all the box-plots.

In Tab. 4.1, we show the median (across the repetitions) of the best non-scaled non-linearity discovered by GP for the most promising tree structures. Here, the balancedness-preserving strategy is employed.

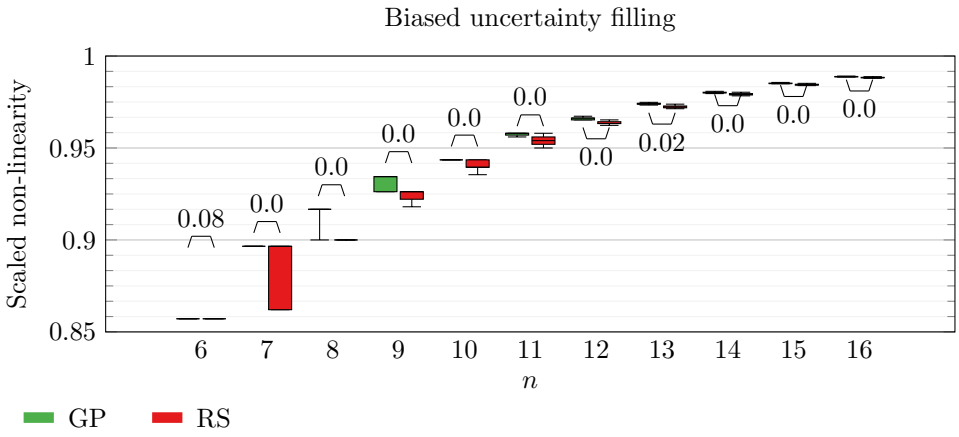

Figure 4.3: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N}}$.

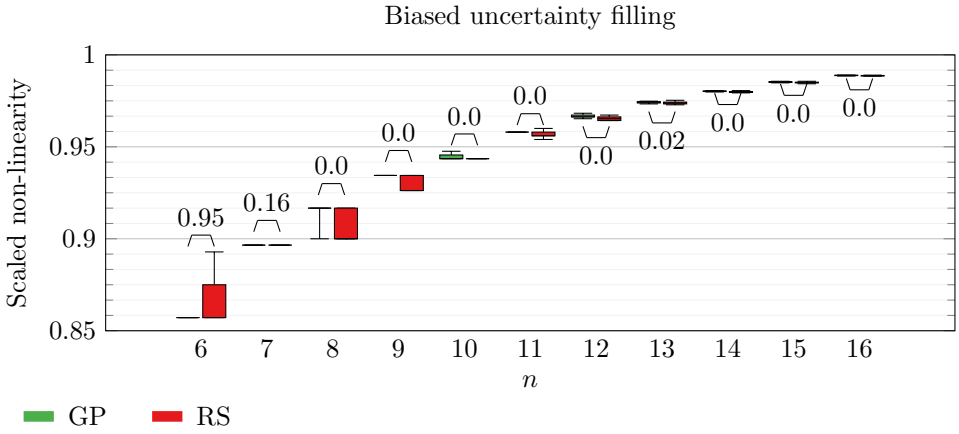

Figure 4.4: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N,ERC}}$.

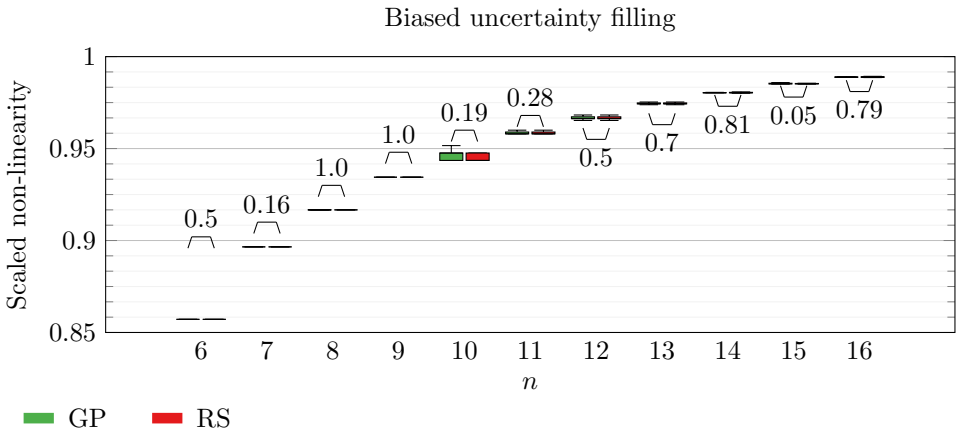

Figure 4.5: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F1}}^{\mathsf{P}}$.

Table 4.1: Table that details the median best non-linearity discovered by GP.

n	6	7	8	9	10	11	12	13	14	15	16
$\mathcal{S}_{\mathsf{F1}}^{\mathsf{N,ERC}}$	25	52	110	228	468	958	1948	3946	7967	16 050	32 276
$\mathcal{S}_{\mathsf{F1}}^{\mathsf{N}}$	25	52	110	228	468	958	1947	3946	7966	16 048	32 274

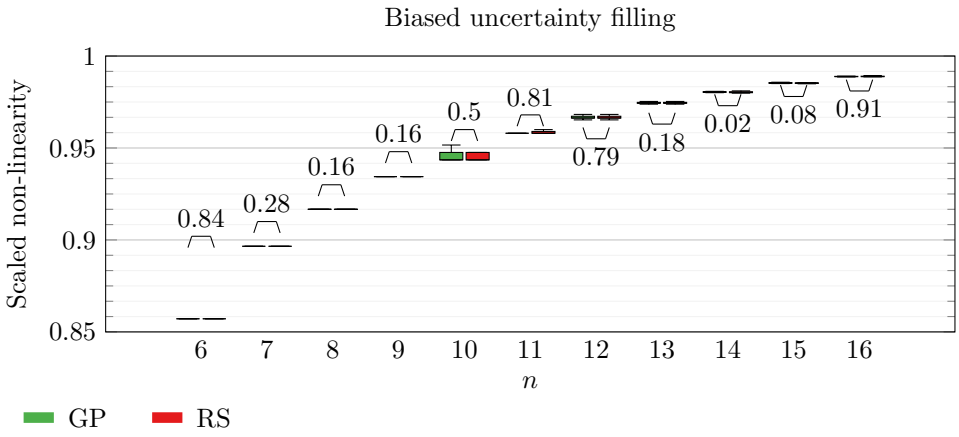

Figure 4.6: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F1}}^{\mathsf{P,ERC}}$.

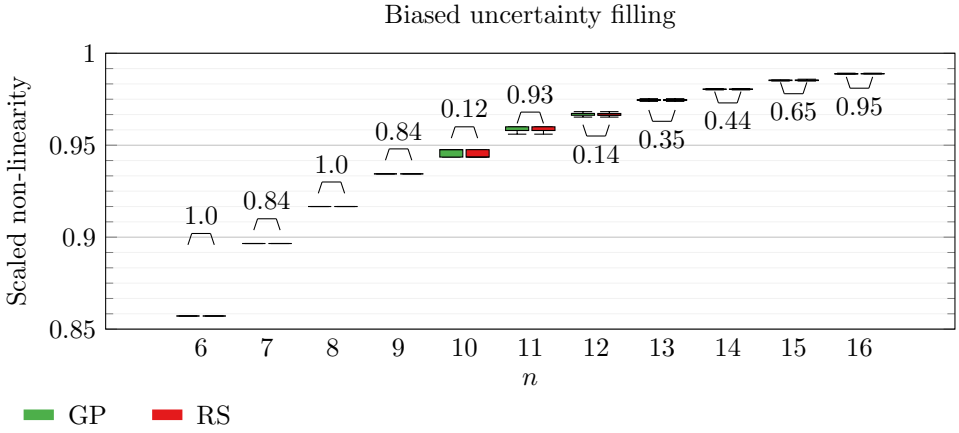

Figure 4.7: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F2}}^{\mathsf{B}}$.

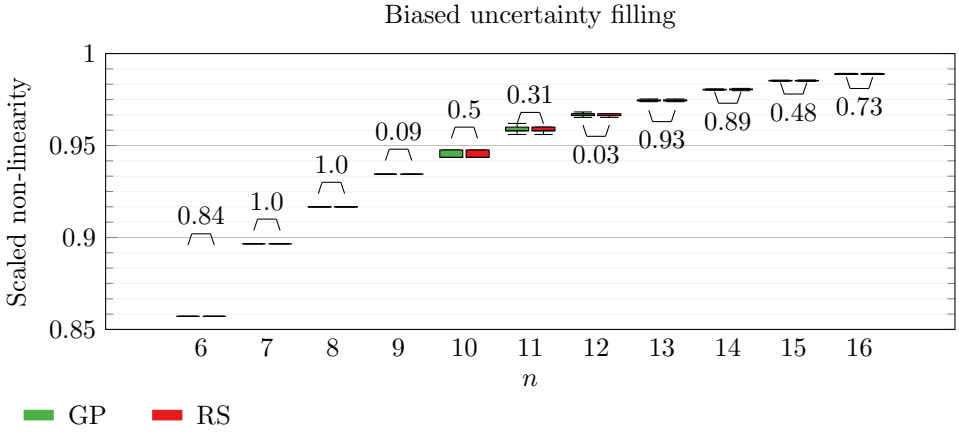

Figure 4.8: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F2}}^{\mathsf{B,ERC}}$.

This analysis proved that GP is better than RS when the same computational budget is employed. This is sufficient to prove that an evolutionary approach is capable of providing a higher-quality search space exploration compared to a purely random approach. This finding should enable practitioners to solely focus on implementing and comparing different evolutionary methods as regards non-linearity maximization.

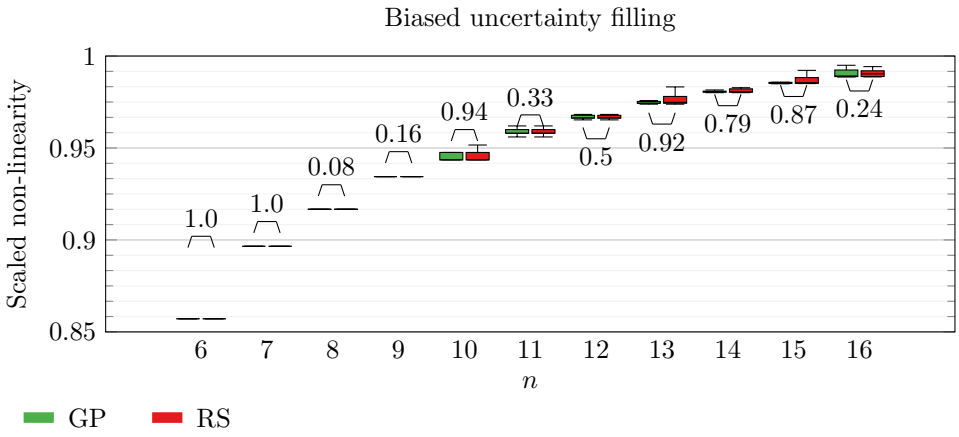

Figure 4.9: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F2}}^{\mathsf{N}}$.

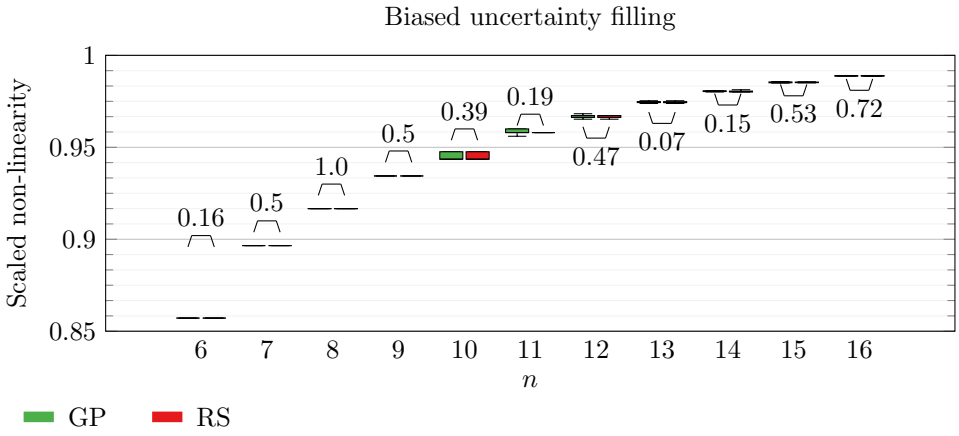

Figure 4.10: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F2}}^{\mathsf{N,ERC}}$.

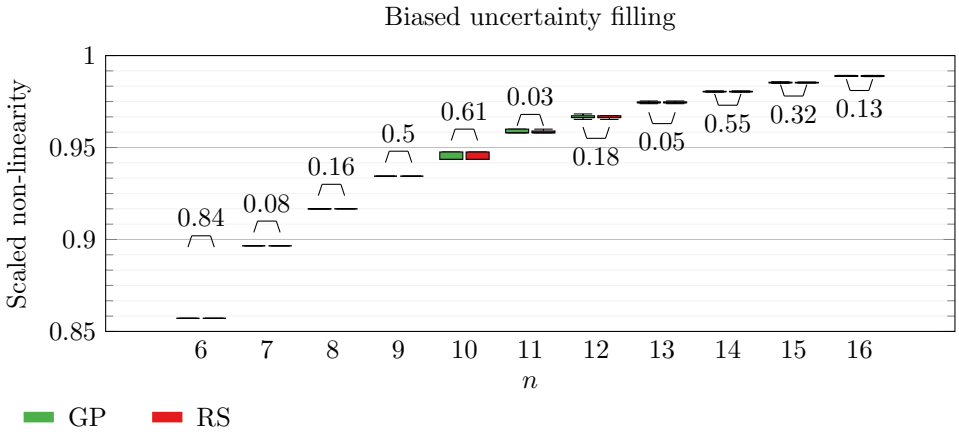

Figure 4.11: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F2}}^{\mathsf{P}}$.

4.4.2 Balancedness

We analyze the unbalancedness degree of the best individuals identified with GP. We perform a Wilcoxon signed-rank test [293] and, for each n, we check whether the biased strategy performs better than the unbiased one in a statistically significant way as regards discovering balanced Boolean functions.

We perform this experiment with $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N,ERC}}$ and $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N}}$, which are the only configurations

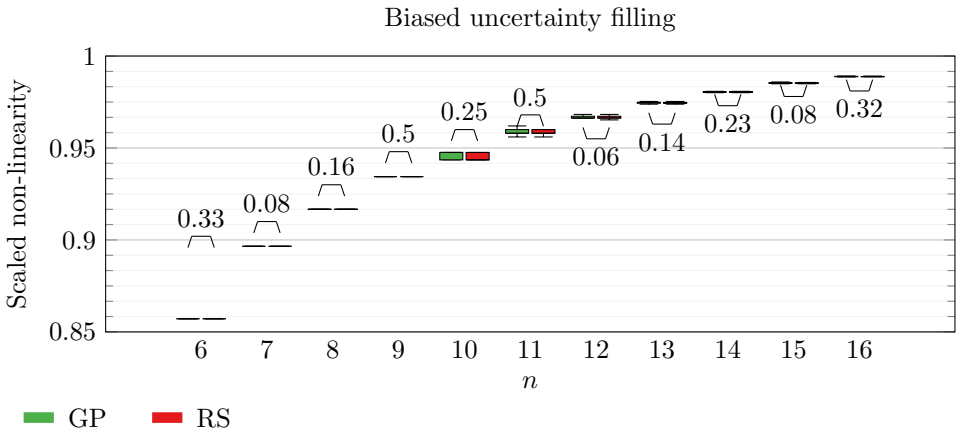

Figure 4.12: Box-plot with comparison between GP and RS with $\mathcal{S}_{\mathsf{F2}}^{\mathsf{P,ERC}}$.

that are promising. The results are shown in Tab. 4.2 and Tab. 4.3.

Table 4.2: Table that details the mean (μ) and standard deviation (σ) of the unbalancedness degree belonging to the best individuals. The p-values of the Wilcoxon test [293] are detailed as well. The results are based on GP with $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N,ERC}}$ as tree structure.

n	Biased	Unbiased	p-value
	$\mu \pm \sigma$	$\mu \pm \sigma$	
6	0.53 ± 2.0	7.53 ± 2.12	0.0
7	0.0 ± 0.0	9.86 ± 5.34	0.0
8	0.0 ± 0.0	15.67 ± 6.95	0.0
9	0.0 ± 0.0	24.03 ± 10.42	0.0
10	0.0 ± 0.0	34.03 ± 19.28	0.0
11	0.0 ± 0.0	37.96 ± 21.88	0.0
12	0.0 ± 0.0	62.84 ± 44.66	0.0
13	0.0 ± 0.0	78.07 ± 42.94	0.0
14	0.0 ± 0.0	94.27 ± 67.75	0.0
15	0.0 ± 0.0	137.8 ± 107.01	0.0
16	0.0 ± 0.0	278.87 ± 185.31	0.0

These results tell us that with the biased uncertainty filling strategy we have a high probability to discover balanced solutions. Given the importance of the balancedness as security feature, we believe that using a biased uncertainty filling strategy is almost mandatory when running optimization algorithms based on spectral inversion. Quite the opposite, with the unbiased uncertainty filling strategy we would probably evolve unbalanced functions.

4.4.3 Uncertainty Positions

We analyze the obtained results to provide a justification to the negative results highlighted in Sec. 4.4.1. We recall that when applying spectral inversion to a symbolic formula we obtain a pseudo-Boolean function represented by its pseudo-truth table in polar form, where negative values are mapped to *True* and positive values are mapped to *False*. Zero values are "uncertain" positions that need to be

Table 4.3: Table that details the mean (μ) and standard deviation (σ) of the unbalancedness degree belonging to the best individuals. The p-values of the Wilcoxon test [293] are detailed as well. The results are based on GP with $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N}}$ as tree structure.

n	Biased	Unbiased	p-value
	$\mu \pm \sigma$	$\mu \pm \sigma$	
6	0.0 ± 0.0	7.03 ± 2.36	0.0
7	0.0 ± 0.0	9.89 ± 4.07	0.0
8	0.0 ± 0.0	17.47 ± 8.2	0.0
9	0.0 ± 0.0	26.22 ± 11.89	0.0
10	0.0 ± 0.0	34.8 ± 17.59	0.0
11	0.0 ± 0.0	46.48 ± 29.98	0.0
12	0.0 ± 0.0	50.8 ± 38.46	0.0
13	0.0 ± 0.0	95.2 ± 69.43	0.0
14	0.0 ± 0.0	101.38 ± 60.62	0.0
15	0.0 ± 0.0	178.51 ± 128.97	0.0
16	0.0 ± 0.0	304.38 ± 218.25	0.0

filled with a random criterion, and in case their percentage is high, then the nearest Boolean function is almost completely calculated randomly.

This invalidates the fitness optimization since we lose the advantages of an actual evolution and we resemble to a standard random search-based approach that does not provide a smarter search space exploration.

Building on this, we analyze the mean uncertainty positions percentage of the pseudo-truth tables corresponding to the individuals in the population across the generations for $n \in \{8, 12, 16\}$. We use the biased uncertainty filling strategy. The trend of the mean uncertainty positions percentage is shown in Fig. 4.13, where the shaded area represents variance. We recall that the binary input string can be encoded in the following ways, resulting in different terminal sets:

1. A terminal set of one variable representing an integer between 0 and $2^n - 1$, i.e., the integer encoded by a given binary string of size n (N);
2. A terminal set of n variables representing the binary positions of the input string (B);
3. A terminal set of n variables representing the binary positions of the input string in polar form (P).

In almost all the cases the mean uncertainty positions percentage is approximately 1.0 for the whole evolution, meaning that the method is equivalent to a random search-based technique in those cases. However, the trend is lower only for $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N,ERC}}$ and $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N}}$, which are the only tree structures that have been shown to outperform RS. Hence, using configurations that differ from the aforementioned ones would lead to the discovery of pseudo-Boolean functions with an extremely high percentage of uncertain positions populated with non-deterministic choices that invalidate the advantages of the evolution. $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N,ERC}}$ and $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N}}$ appear to be the only structures that discover pseudo-Boolean functions that can be improved through a GP-based evolution, since the percentage of uncertain positions is low enough in a way that

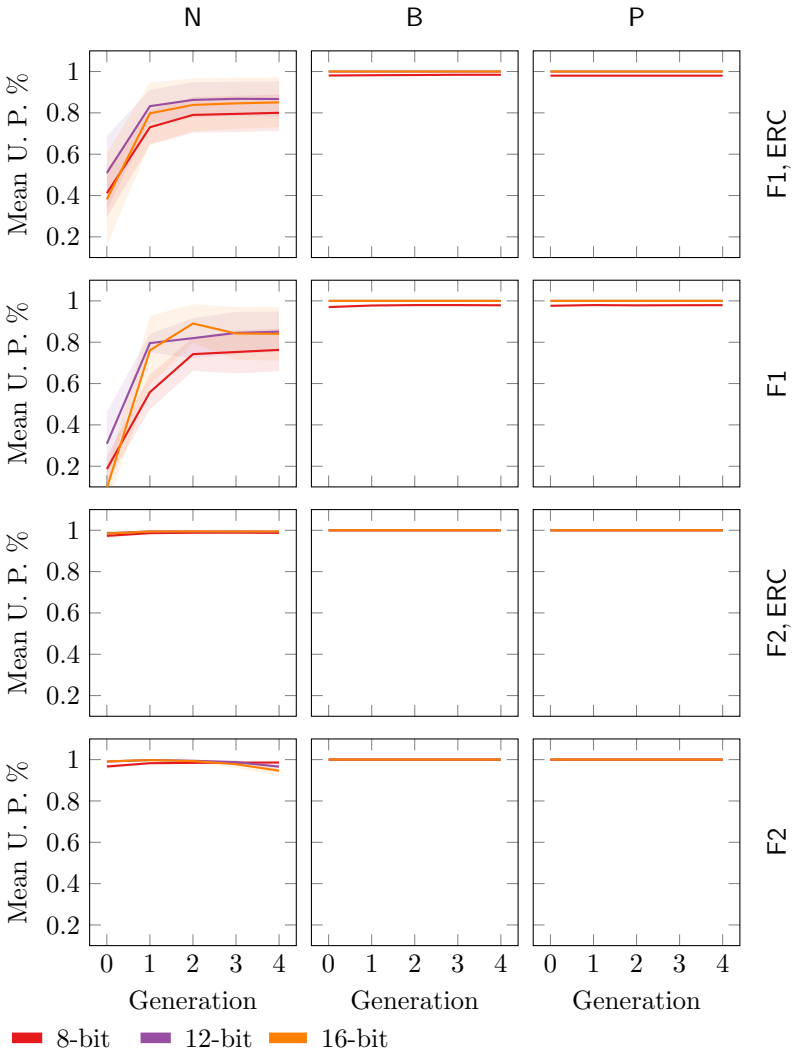

Figure 4.13: Trend of the mean uncertainty positions percentage across the generations.

the individual is not completely randomly built.

4.5 Discussion

We show that the proposed biased uncertainty filling strategy enables the discovery of balanced solutions through GP. Achieving balanced functions is crucial because unbalanced functions introduce a statistical bias that could be exploited by attackers. Thus, ensuring a high likelihood of generating balanced solutions is vital when applying a spectral inversion technique.

We demonstrate that even with a small population size and a limited number of generations, GP can discover superior solutions compared to RS, provided a specific type of symbolic tree structure is used. In this scenario, the (scaled) non-linearity often exceeds 0.90. This indicates that by employing tree structures like $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N,ERC}}$ and $\mathcal{S}_{\mathsf{F1}}^{\mathsf{N}}$ in conjunction with GP and biased uncertainty filling, the proposed method can identify Boolean functions with high non-linearity. Thus, exploring the search space formed by Walsh transforms appears to be a promising strategy. However, it is important to note that, for n where optimal non-linearity has been extensively

studied and derived [73], GP is still not capable of discovering Boolean functions with such non-linearity value.

As an additional note, the two aforementioned tree structures are the only ones, among the examined, that exhibit a low percentage of uncertain positions in the pseudo-truth tables evolved across the generations, enabling the optimization to take advantage from the GP-based evolution to reach better solutions.

This work offers an in-depth exploration of designing and implementing Boolean functions as components for secure stream ciphers. Creating a secure Boolean function is crucial for developing a confidentiality-safe stream cipher. By utilizing a Vernam cipher with a PRG, the need for a key as long as the message is eliminated, which is essential for practical use in real-world scenarios. Furthermore, the Combiner model can be used as a PRG to ensure communication confidentiality, but this is only effective if the Boolean function provided to the Combiner model exhibits strong security properties.

Our proposed method enables the construction of such Boolean functions with the desired security characteristics. By executing our evolutionary process, selecting the optimal solution, and integrating it into a Combiner model used as a PRG in a Vernam cipher, we achieve a confidentiality-safe stream cipher suitable for real-world applications, effectively securing communications from eavesdroppers.

Chapter 5

Large Language Models Code Generation Enhancement via Genetic Improvement

Contents

In the context of program synthesis and software engineering, we explore the potential of improving the code generated by Large Language Models (LLMs) through the use of Grammatical Evolution (GE) [15], a particular Genetic Programming (GP) method [5]. This technique is defined as Genetic Improvement (GI) [153, 154]. Here, the GE process is initialized with solutions provided by an LLM, which serve as a starting point for further refinement. This hybrid approach seeks to combine the strengths of both LLMs and Evolutionary Algorithms (EAs) to provide a more correct version of the solution provided by the LLM alone according to a set of user-provided test cases. This work is an extension of [169].

5.1 Background

Software development is a complex endeavor requiring specialized skills, knowledge, and expertise. Consequently, the creation of tools to assist programmers has always been a key focus in the field. Recently, particular types of Transformer-based architectures [130] defined as Large Language Models (LLMs) have been effectively utilized for automatic code generation in tools like Copilot [139, 145]. The release of Transformers and LLMs revolutionized the field of Natural Language Processing (NLP) since these models were trained on large text corpora to improve their capability to predict missing text given an input text.

Meta AI and OpenAI are the companies that, in recent years, developed and released to the public some of the most effective LLMs available nowadays.

In 2023, Meta AI launched the first generation of LLaMA [133], a series of open-source pre-trained LLMs ranging from 7 to 65 billion parameters, aimed at supporting research and development. Alpaca [294] was developed by Stanford University as a fine-tuned version of LLaMA using the self-instruct method for instruction tuning [295]. It was specifically trained on instruction-following demonstrations. In 2023, an upgraded version, LLaMA 2 [134], was introduced, featuring up to 70 billion parameters. The following year, a code-specialized variant called CodeLLaMA [296] was released, and in 2024, the next iteration in the LLaMA family, LLaMA 3 [135], made its debut.

OpenAI has also made significant strides in the development of LLMs with GPT-3 [136] and its successors, GPT-3.5 and GPT-4 [137]. These models, based on decoder-only Transformer architectures, vary in their number of parameters and the scale of their training data. GPT-4, available through ChatGPT Plus and OpenAI APIs, continues to deliver strong performance across a variety of text-based tasks, but the specifics of its training data and methodologies are not publicly disclosed. In 2024, CriticGPT [297], a GPT-4-based model designed to identify errors in code generated by ChatGPT, was also released.

Currently, LLMs are capable of generating function and method implementations based on problem descriptions. However, when faced with poorly defined or inherently complex problems, LLMs may produce flawed or incomplete code. Despite inaccuracies, such code often provides a foundational structure that can serve as a starting point for further development and refinement.

5.2 Problem Formulation

Given a coding problem Q defined by a textual description and a set of test cases—each being a single valid input-output pair provided by the user—defined as $D_Q = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in X, \mathbf{y}_i \in Y, i \in [1..N]\}$ with $|D_Q| = N$, where X denotes the input space and Y denotes the output space, we aim to create a program M that correctly implements the problem description and produces the expected output for each provided input.

In our scenario, we start by crafting a solution $\tilde{M}$ by prompting an LLM with Q. Then, the LLM-provided solution $\tilde{M}$ is improved and refined through an evolutionary process to enhance its correctness w.r.t. D_Q and, finally, a new program M is built as a result of the evolutionary optimization.

5.3 Proposed Method

In our approach, the textual description is fed into the LLM as input (i.e., prompt), while the test cases are used to define the fitness function. The proposed method is divided into three macro-steps, where the latter two compose GI:

1. **Extraction of the LLM output**: The LLM is provided with the problem description, and the generated code is extracted and processed. This processed code can then be potentially transformed into a genome starting from an

Abstract Syntax Tree (AST);

2. **Dynamic grammar definition and solution encoding**: Both the problem description and the initial solutions generated by the LLM are examined to identify additional problem-specific functions, variables, and constants that can be used to tailor a base grammar for the chosen programming language. The LLM-generated solutions are then parsed from the AST representation into genomes according to this specialized grammar;
3. **Evolution**: GI is initiated and executed using the LLM-derived solution as a starting point, with a fitness function that considers both the number of failed test cases and the associated errors.

5.3.1 Extraction of the LLM Output

Even with a well-crafted prompt, the output from an LLM may contain unnecessary text. Therefore, the first step involves isolating the code from the LLM response. This extraction process varies depending on the specific coding language and the specific LLM used, as different models may format or delimit code differently.

In our approach, the code extraction process must be designed to retrieve both the code elements that remain unchanged by GI (such as import statements and auxiliary functions or classes) and the elements that will be refined by GI (such as the primary function or class).

It is important to recognize that the extracted code may not always be syntactically correct and, therefore, cannot be directly encoded into an AST for use in GI. To prevent solutions with syntax errors from being discarded prematurely, we implement the following heuristic to attempt to generate a syntactically correct program:

1. We extract the index of the first line with at least one syntax error;
2. We remove the corresponding line;
3. We repeat step 1 and step 2 until no syntax error exists or until no code remains.

This heuristic is important since it enables the entire pipeline to be actually executed (the pipeline needs as strict requirement to work with a syntactically valid program). However, different heuristics can be implemented depending on the coding language. We believe that the presented heuristic is general enough to be applied regardless of the coding language and, moreover, it is designed to be as less disruptive as possible, focusing on only removing the lines containing syntax errors. If the program has a few syntax error, then the impact of this heuristic on the pipeline should be limited. However, if the program is deeply syntactically wrong, then this heuristic may invalidate the program and bring the entire pipeline to build a new program from scratch. Coding language-specific heuristics should focus on smartly fixing the syntax errors without removing entire code lines.

Comments are removed and only existing import statements are retained. All variables and function parameters are renamed as v_i with i ranging from 0 to the number of variables minus one, since their semantic names are immaterial as regards code correctness.

Finally, we obtain a syntactically valid code that can be encoded into an AST.

5.3.2 Dynamic Grammar Definition and Solution Encoding

Since the Backus-Naur Form (BNF) grammar for a language such as Python, the one adopted in our experiments, can be complex and large, we only parse a subset of it, which contains the most common use cases and constructs so that the size of the grammar remains relatively small.

Base grammar

Let $\mathcal{G}$ denote a BNF grammar that encapsulates the core constructs of the programming language used to address the problem. It is important to highlight that $\mathcal{G}$ might deviate from the standard formal grammar for the language, incorporating specific biases and modifications tailored for the evolutionary process.

Firstly, $\mathcal{G}$ is structured to favor the creation of small, concise programs. This is typically achieved by adding extra derivations to expansions that include terminal symbols rather than non-terminals, thereby skewing random selection towards shorter programs without changing the language that is recognized or generated. Additionally, the grammar can be customized to produce code that already incorporates frequently used libraries and functions, avoiding the need to generate their names from scratch.

Dynamic Grammar Definition

To support the evolutionary process, we utilize the information from both the problem textual description and the solutions generated by the LLM to refine the grammar, integrating problem-specific constants and symbols. Specifically, we integrate $\mathcal{G}$ with additional terminal symbols, which are constants and function names.

It can be difficult to discover during the evolution constants appearing the problem description (e.g., a task involving "finding all multiples of 15 less than 100" includes two constants). Quite the contrary, including these constants directly from the problem description simplifies their usage in the solution. In a similar way, including functions and libraries can prevent the need to recreate existing code from scratch. However, identifying useful libraries is a complex task that requires an understanding of the language ecosystem and not just its grammar.

LLMs are a useful source of information on which libraries can be useful, being trained on large corpora of text including code. Given the LLM output and the problem description, we present a general procedure to extract the needed information:

1. We extract the imported libraries, function names, strings, and numbers from the LLM output to ensure that libraries and functions that are useful for building the solution can be directly selected during the evolution;
2. We extract strings and numbers from the problem description as constants;
3. We extract keywords from the problem description as strings by leveraging KeyBERT [298].

A problem-specific grammar $\mathcal{G}_Q$ can be created by enhancing $\mathcal{G}$ with the extracted symbols, including all variable names as terminal symbols. To increase the likelihood

of these extracted symbols being used in a solution, the corresponding derivations can be duplicated multiple times, making them more likely to be selected.

This approach allows the evolutionary process to leverage existing library functions and avoids the need to generate constants from scratch when they can be derived from the problem description. However, the effectiveness of this grammar is highly dependent on the quality of the LLM output and the provided problem description. If the output is incorrect but not in a fundamental way and it includes relevant library function calls, it can guide the search effectively. On the other hand, if the output is far from a viable solution, it can introduce irrelevant constants and functions that may hinder the evolutionary process.

Once the grammar $\mathcal{G}_Q$ is defined, the LLM-provided solutions can be converted from the AST into genomes for use in the evolutionary process.

5.3.3 Evolution

Since GI is employed to refine existing solutions, two key aspects need to be determined: the method for generating the initial population and the approach for evaluating individuals. Other evolutionary process parameters, such as the choice of genetic operators, are standard and are not specific to the proposed method.

The initial population is seeded with solutions generated by the LLM, encoded as previously described, with potential repetitions if the number of solutions is less than the population size. This replication strategy helps conserve computational resources, especially when using larger LLM models, which can be resource-intensive.

In case of LLMs with less than 7 billions parameters, then a low-budget GPU, such as a Nvidia A100, should enable the LLM to output a solution in less than 30 seconds. However, larger LLMs, with more than 20 billions parameters, require more powerful GPU and potentially more time to compute a response locally. Moreover, if a proprietary LLM is employed, such as the ones from OpenAI, then querying the LLM can also cost money, with the exact amount depending on the pricing policy of the LLM APIs. Using this strategy enables the entire pipeline to be executed even though the available budget, both computational and economic, is low.

Fitness function is defined starting from the set of test cases D_Q provided by the user. The fitness value consist of two components: the first ($\mathcal{F}$) considers the number of not passed test cases, while the second ($\mathcal{E}$) assesses the distance between the expected outputs and the generated outputs. These two components are combined using a lexicographic ordering ($\mathcal{F}_\mathcal{E}$) that prioritizes the $\mathcal{F}$ component. Given D_Q with size N, a problem Q, and a program M, the first component is defined as:

$$\mathcal{F}(M, D_Q) = \sum_{i=1}^{N} [\text{out}(M, \mathbf{x}_i) \neq \mathbf{y}_i] \tag{5.1}$$

where $\text{out}(M, \mathbf{x}_i)$ represents the output of M on input $\mathbf{x}_i$ and $[\cdot]$ is the indicator function, which is 1 if the enclosed proposition is true, 0 otherwise.

The second component is defined as:

$$\mathcal{E}(M, D_Q) = \sum_{i=1}^{N} \text{dist}(\text{out}(M, \mathbf{x}_i),\ \mathbf{y}_i) \tag{5.2}$$

where dist is a measure of dissimilarity that depends upon the type of the provided values. The solutions are ordered according to the number of not passed test cases and, in case of tie, according to how close they are to the correct output.

5.4 Experimental Analysis

We test four LLMs: CodeLLaMA 7B (`CL7`), LLaMA 3 8B (`L3`), ChatGPT (`CG`), and GPT4 (`G4`).

5.4.1 Language

While the proposed approach has been described in general terms, our experiments were specifically focused on the Python language. This decision was influenced by the extensive training resources available for widely-used languages, which improve the LLM ability to generate effective solutions.

In the experiments, the problem description was prefaced with the following text when prompting the LLM:

```
Write a single Python function to solve the following problem
and insert the necessary modules:
```

We assume that the LLM generates a solution within a single function that can be executed by passing one or more arguments as inputs and returning one or more outputs.

Given that an LLM response might include multiple functions, we assume the last function is the main one, with earlier functions serving as support functions likely called within the main function. This assumption is based on the common programming practice of placing the main function after any supporting functions.

Our focus is on evolving only the body and parameters of the main function, keeping all import statements and supporting functions unchanged to avoid expanding the search space unnecessarily.

The process of extracting code from the LLM responses begins with removing all comments and extracting valid import statements and functions. Among these, the last function is designated as the main one, while the others are retained if they are syntactically correct. The body of the main function is then adjusted to be syntactically correct using the heuristic described in Sec. 5.3.1. If the resulting body is empty, it is replaced with `pass`.

5.4.2 Grammar

The grammar employed is a subset of the full Python grammar, briefly summarized here. The grammar starts by generating a call to the main function, which accepts several parameters as inputs and contains a body with executable code. The function body may include one or more statements, which are divided into conditional (e.g., if, for, while) and non-conditional types.

Non-conditional statements can include return, assignment, print, pass, continue, break, raise, or invoking an instance method on an object. A node represents a value that can be a string (possibly formatted), a number, a Boolean, a problem

parameter, null, a logical or mathematical combination of two variables, a list, a tuple, a list comprehension, a tuple comprehension, or a function applied to variables. The full base grammar in BNF format for the Python subset we use can be found in our repository (see Sec. 5.4.5).

While it would be possible to parse all problems by defining a grammar that encompasses the entire Python language, doing so would significantly increase the search space, potentially slowing down the evolutionary process. Therefore, the grammar used seeks to strike a balance between expressiveness and search space complexity.

5.4.3 Fitness

The dissimilarity measure adopted in $\mathcal{E}$ depends upon the types of the variables that are involved in the comparison:

- Numeric values (integers, floats, and Booleans) are compared with the absolute difference;
- Strings, lists, and ranges are compared with the edit distance;
- Sets and dictionaries are compared with the Jaccard distance;
- Tuples are compared by summing the dissimilarities of the elements contained in them.

Since the code may return a value of a different type, we manage this scenario by assuming the dissimilarity to be the maximum dissimilarity possible on the training test cases assuming as output a "neutral" element (i.e., 0 for numbers, the empty string or list, and the empty set or dictionary). If the code raises an exception during its evaluation that value is doubled.

It is important to note that tuples are not directly compared using edit distance as with other collection-like data structures. This decision is based on the fact that when a problem requires a function to return multiple outputs, these outputs are typically bundled into a tuple, which is the default behavior in Python. As a result, applying edit distance directly to the tuple could yield a measure of dissimilarity that is too coarse to be effective.

5.4.4 Problems

To evaluate our approach, we use the widely-recognized problem dataset PSB2 [143, 144] (Tab. 5.1), which offers a comprehensive set of problems, making it an excellent benchmark for testing automatic code generation algorithms. For each problem, we prompt the LLM to generate a solution, repeating this process independently 10 times with the same prompt. If the LLM successfully solves the problem (i.e., the generated code is correct based on the provided training examples) in at least one of these attempts, we exclude it from further refinement. Problems solved by the LLM are marked with a ✓ in the following tables.

Because of the importance of the problem description when prompting an LLM, we adopt the official problem descriptions available in the PSB2 paper itself [143, 144].

Table 5.1: List of PSB2 problems.

Name	Alias	Name	Alias
Basement	BS	Bouncing Balls	BB
Bowling	BW	Camel Case	CC
Coin Sums	CS	Cut Vector	CV
Dice Game	DG	Find Pair	FP
Fizz Buzz	FB	Fuel Cost	FC
Greatest Common Divisor	GD	Indices of Substring	IS
Leaders	LD	Luhn	LH
Mastermind	MM	Middle Character	MC
Paired Digits	PD	Shopping List	SL
Snow Day	SD	Solve Boolean	SB
Spin Words	SW	Square Digits	SQ
Substitution Cipher	SC	Twitter	TW
Vector Distance	VD		

5.4.5 Results

For each tested LLM, we prompt it with a given problem description 10 times, leading to 10 solutions. These solutions are uniformly replicated to meet the required population size, which is 200. We run each evolution for 100 generations. We avoided incrementing the computational budget to force the evolution to run for a short amount of time, which is a more probable setting in real cases. The fitness is computed by using 1000 test cases as training data and 1000 test cases as test data. The fitness is computed on 50 training test cases sampled without replacement at the beginning of each generation during the execution of the selection procedure [299, 300, 301]. Preliminary experiments showed that this modification does not meaningfully impact the quality of the final solutions, providing a considerable speed-up during the selection phase. Moreover, we verified that a sample of 50 is a correct trade-off between speed-up and reliability of the results.

For each combination of problem and LLM, we perform 10 separate GI runs starting from the same initial population. The test case datasets are retrieved via the psb2 library [143, 144]. We set an initial maximum depth of 15 and a maximum depth of 30 for the individuals.[1] We do not impose a wrap-around limit for the genomes and we set a maximum codon size of 200.

We test two selection algorithms: tournament selection with a tournament size of five (T) and lexicase selection (L) [302, 303, 304, 305]. Both crossover (applied with probability 0.80) and mutation (applied with probability 0.60) are sub-tree operators. We chose to increase the mutation probability w.r.t. the standard values that are usually employed in GP to force the evolution to modify programs and avoid having many similar solutions. An individual applied to a set of 1000 test cases is associated with a time-out of three seconds to avoid to stuck the evolution with excessively inefficient solutions and non-terminating ones (e.g., with infinite loops).

We implement a method based on the technique described in [166] to test the

[1]The BW problem requires a maximum depth of 40 since the initial solution has a large size.

self-correction capabilities of LLMs. For each LLM, we initially prompt the LLM asking for a solution of a given problem and then we evaluate the correctness of the generated code (strategy defined as $\mathcal{L}$). If the code is incorrect, we prompt the LLM to self-correct its own output (strategy defined as $\mathcal{L}^+$).

During a self-correction session, we ask the LLM to iteratively improve the output code up to 10 times:

- If in a given iteration, the time-out is reached, then we prompt the LLM with:

  ```
  Your code is too slow. Please, rewrite it and make it
  correct and more efficient. Remember to insert the necessary
  modules.
  ```

- If in a given iteration, the code contains at least one syntax error, then we prompt the LLM with:

  ```
  Your code contains syntax errors. Please, rewrite it and
  fix all the syntax errors. Remember to insert the necessary
  modules.
  ```

- If the code is semantically incorrect, we provide 10 pairs of input-output examples from the training data that were not passed in the previous iteration. Hence, we prompt the LLM with:

  ```
  Your code is incorrect. Please, rewrite it. Remember
  to insert the necessary modules. Make sure that
  ```

  followed by the examples formatted as `input -> output`.

We perform two self-correction runs. For each run, we store the best overall solution across the iterations according to the number of passed test cases on the training set.

We execute our GI runs on a virtual machine with Ubuntu 20.04, with 16 dedicated cores on an Intel(R) Xeon(R) Gold 6140 CPU, and 128 GB RAM. We measure that, on average, a single repetition of an evolutionary process executed for 100 generations with a population size of 200 takes less than 30 minutes, considering the fitness evaluations on both the training and test set. Our code is available at https://github.com/dravalico/LLMGIpy. In our implementation, to perform the GE part, we use the ponyge2 library [160].

Statistical significance is assessed by using a Wilcoxon-Mann-Whitney test [306] with $\alpha = 0.05$. When the number of comparisons is greater than two, a preliminary Kruskal-Wallis test [307] with $\alpha = 0.05$ is performed and then, if there is a significant statistical difference among the methods in the group, a Holm-Bonferroni correction [308] with $\alpha = 0.05$ is applied.

We analyze two types of GI:

- $\mathcal{F}$ in combination with tournament selection ($\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$) as in [169];
- $\mathcal{F}_{\mathcal{E}}$ in combination with lexicase selection ($\mathcal{I}_{\mathsf{L}}^{\mathcal{F}_{\mathcal{E}}}$).

The results are shown in Tab. 5.2 and Tab. 5.3. The tables show the median of passed test cases (scaled in $[0, 1]$) on the test set by the best overall solutions according to the training set for $\mathcal{L}$, $\mathcal{L}^+$ and GI across the executed repetitions. For

each LLM, the table highlights both the problems that are directly solved (✓) and those that are not parsed by the grammar (`np`). We use 0 to indicate when a method does not pass any test case for all repetitions. We use 1 to indicate when a method passes all the test cases in all repetitions.

Table 5.2: Median number of passed test cases (scaled in $[0, 1]$) for each problem and LLM. We highlight in bold the methods that are better for the same problem and LLM in a statistically significant way. The GI is based on $\mathcal{F}$ with tournament selection.

	CL7			L3			CG			G4		
	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI
BS	✓	—	—	✓	—	—	✓	—	—	✓	—	—
BB	0	0	0	0	0	0	0.00	0.02	**0.04**	0.00	0.04	**0.04**
BW	0.00	0	**0.01**	0.00	0.00	**0.09**	0.00	0.01	**0.01**	0.03	0.42	0.11
CS	0	0	0	0	0	0	0	0	**0.00**	0	0.15	0.10
CC	0	0	**0.29**	✓	—	—	✓	—	—	✓	—	—
CV	0	**0.06**	0.00	0.17	0.30	**1**	0.92	0.49	0.99	0.85	0.85	**0.99**
DG	0.00	0	**0.00**	0.00	0.14	**0.14**	0.07	0.07	**0.14**	0.14	0.00	0.14
FP	✓	—	—	0	**0.76**	0	✓	—	—	✓	—	—
FB	✓	—	—	✓	—	—	✓	—	—	✓	—	—
FC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
GD	✓	—	—	✓	—	—	✓	—	—	✓	—	—
IS	0.78	0.78	0.78	✓	—	—	✓	—	—	✓	—	—
LD	0.22	0.22	**0.34**	0.58	1	1	✓	—	—	✓	—	—
LH	0	0	**0.04**	✓	—	—	✓	—	—	✓	—	—
MM	0.06	0.06	**0.17**	0.10	0.14	**0.71**	✓	—	—	✓	—	—
MC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
PD	0	**1**	0.48	✓	—	—	✓	—	—	✓	—	—
SL	0	0	0	0	0	0	0	0	0	0	**0.50**	0
SD	0.04	0.04	0.04	0.04	0.02	0.04	0.04	0.04	**0.75**	0.04	0.04	**0.08**
SB	0	0	`np`	0.00	0	**0.50**	0.00	0.62	**0.50**	0	**0.68**	0.50
SW	✓	—	—	0.35	0.67	**0.37**	✓	—	—	✓	—	—
SQ	0	0	0	✓	—	—	✓	—	—	✓	—	—
SC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
TW	✓	—	—	✓	—	—	✓	—	—	✓	—	—
VD	0	0	0.00	✓	—	—	✓	—	—	✓	—	—

`CL7` solves a smaller number of problems and $\mathcal{L}^+$ appears to be ineffective and does not offer any improvement except for PD (where it reaches an optimum across all repetitions) and CV. In the latter case, only when compared to $\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$ the self-correction $\mathcal{L}^+$ is better in a statistically significant way. OpenAI models directly solve most of the problems. $\mathcal{L}^+$ is better than the two compared strategies in a statistically significant way in only 3 cases: FP for `L3`, and SL and SB for `G4`. Therefore, it appears that simply re-prompting the LLM usually does not lead to any meaningful improvement. Many problems that are not directly solved by the LLM show meaningful improvements with GI, where $\mathcal{I}_{\mathsf{L}}^{\mathcal{F}\varepsilon}$ provides a slightly higher number of them. Our GI method can thus enhance LLM-generated code and it is never worse than the LLM alone. However, the optimum is obtained in only 2 cases

Table 5.3: Median number of passed test cases (scaled in $[0, 1]$) for each problem and LLM. We highlight in bold the methods that are better for the same problem and LLM in a statistically significant way. The GI is based on $\mathcal{F}_{\mathcal{E}}$ with lexicase selection.

	CL7			L3			CG			G4		
	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI
BS	✓	—	—	✓	—	—	✓	—	—	✓	—	—
BB	0	0	0.00	0	0	**0.00**	0.00	0.02	**0.04**	0.00	0.04	**0.04**
BW	0.00	0	**0.01**	0.00	0.00	**0.09**	0.00	0.01	**0.03**	0.03	0.42	0.10
CS	0	0	0	0	0	0	0	0	**0.02**	0	0.15	**0.62**
CC	0	0	**0.29**	✓	—	—	✓	—	—	✓	—	—
CV	0	0.06	**0.35**	0.17	0.30	**1**	0.92	0.49	0.99	0.85	0.85	**1.00**
DG	0.00	0	**0.01**	0.00	0.14	**0.14**	0.07	0.07	**0.14**	0.14	0.00	0.14
FP	✓	—	—	0	**0.76**	0.00	✓	—	—	✓	—	—
FB	✓	—	—	✓	—	—	✓	—	—	✓	—	—
FC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
GD	✓	—	—	✓	—	—	✓	—	—	✓	—	—
IS	0.78	0.78	0.78	✓	—	—	✓	—	—	✓	—	—
LD	0.22	0.22	**0.52**	0.58	1	1	✓	—	—	✓	—	—
LH	0	0	**0.04**	✓	—	—	✓	—	—	✓	—	—
MM	0.06	0.06	**0.17**	0.10	0.14	**0.72**	✓	—	—	✓	—	—
MC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
PD	0	**1**	0.79	✓	—	—	✓	—	—	✓	—	—
SL	0	0	0	0	0	0	0	0	0	0	**0.50**	0
SD	0.04	0.04	0.04	0.04	0.02	**0.09**	0.04	0.04	**0.75**	0.04	0.04	**0.12**
SB	0	0	`np`	0.00	0	**0.50**	0.00	0.62	**0.50**	0	**0.68**	0.00
SW	✓	—	—	0.35	0.67	**0.38**	✓	—	—	✓	—	—
SQ	0	0	0.00	✓	—	—	✓	—	—	✓	—	—
SC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
TW	✓	—	—	✓	—	—	✓	—	—	✓	—	—
VD	0	0	**0.00**	✓	—	—	✓	—	—	✓	—	—

for $\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$ and 3 cases for $\mathcal{I}_{\mathsf{L}}^{\mathcal{F}\mathcal{E}}$. Moreover, the grammar fails to provide at least one parsable solution only in SB for `CL7`.

We show the comparison between $\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$ and $\mathcal{I}_{\mathsf{L}}^{\mathcal{F}\mathcal{E}}$ in Tab. 5.4. The number of passed test cases is almost similar. Especially, $\mathcal{I}_{\mathsf{L}}^{\mathcal{F}\mathcal{E}}$ is statistically significantly better than $\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$ in some cases (especially for `CL7` and `L3`), while $\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$ outperforms $\mathcal{I}_{\mathsf{L}}^{\mathcal{F}\mathcal{E}}$ only in SB for `G4`. It is interesting to note that with $\mathcal{I}_{\mathsf{L}}^{\mathcal{F}\mathcal{E}}$ we achieve an additional optimum in the median in CV for `G4`.

Ablation Study

With the purpose of checking that the improvement in the number of passed test cases was influenced by both lexicase selection and $\mathcal{F}_{\mathcal{E}}$, we conduct an ablation study by executing the experiments while isolating each of these properties. We analyze the following configurations and we compare them with $\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$ and $\mathcal{I}_{\mathsf{L}}^{\mathcal{F}\mathcal{E}}$:

- $\mathcal{F}$ combined with lexicase selection ($\mathcal{I}_{\mathsf{L}}^{\mathcal{F}}$) in Tab. 5.5;

Table 5.4: Median number of passed test cases (scaled in $[0, 1]$) for each problem and LLM. We highlight in bold the methods that are better for the same problem and LLM in a statistically significant way.

	CL7		L3		CG		G4	
	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$	$\mathcal{I}_{\mathsf{L}}^{\mathcal{F}_{\mathcal{E}}}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$	$\mathcal{I}_{\mathsf{L}}^{\mathcal{F}_{\mathcal{E}}}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$	$\mathcal{I}_{\mathsf{L}}^{\mathcal{F}_{\mathcal{E}}}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$	$\mathcal{I}_{\mathsf{L}}^{\mathcal{F}_{\mathcal{E}}}$
BB	0	0.00	0	**0.00**	0.04	0.04	0.04	0.04
BW	0.01	**0.01**	0.09	0.09	0.01	0.03	0.11	0.10
CS	0	0	0	0	0.00	0.02	0.10	**0.62**
CC	0.29	0.29	—	—	—	—	—	—
CV	0.00	**0.35**	1	1	0.99	0.99	0.99	1.00
DG	0.00	**0.01**	0.14	0.14	0.14	0.14	0.14	0.14
FP	—	—	0	0.00	—	—	—	—
IS	0.78	0.78	—	—	—	—	—	—
LD	0.34	**0.52**	1	1	—	—	—	—
LH	0.04	**0.04**	—	—	—	—	—	—
MM	0.17	0.17	0.71	**0.72**	—	—	—	—
PD	0.48	0.79	—	—	—	—	—	—
SL	0	0	0	0	0	0	0	0
SD	0.04	0.04	0.04	**0.09**	0.75	0.75	0.08	**0.12**
SB	np	np	0.50	0.50	0.50	0.50	**0.50**	0.00
SW	—	—	0.37	**0.38**	—	—	—	—
SQ	0	0.00	—	—	—	—	—	—
VD	0.00	0.00	—	—	—	—	—	—

- $\mathcal{F}_{\mathcal{E}}$ combined with tournament selection ($\mathcal{I}_{\mathsf{T}}^{\mathcal{F}_{\mathcal{E}}}$) in Tab. 5.6.

Considering that $\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$ is the starting configuration implemented and tested in [169], we analyze the impact of lexicase selection alone in Tab. 5.7, and we show that this strategy provides an improvement in the number of passed test cases but the overall contribution is limited. In a similar way, with Tab. 5.8 we show that the integration of $\mathcal{F}_{\mathcal{E}}$ offers marginal or none enhancement at all compared to $\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$. On the other hand, with Tab. 5.9 we show that combining $\mathcal{F}_{\mathcal{E}}$ with lexicase selection leads to an overall increase in the number of passed test cases.

5.5 Discussion

As a preliminary observation, the most advanced LLMs generally produce higher-quality code than their smaller counterparts. However, even these larger models often generate incorrect code for many of the problems under review. When this happens, we re-prompt the LLM with a few failed test cases, asking it to revise the code to pass these tests. Despite these efforts, our findings suggest that if the model initially generates faulty code, it seldom manages to correct itself effectively through re-prompting alone. This underscores the challenges LLMs face in creating accurate code for specific tasks, even when provided with concrete examples, and suggests that additional methods are required to achieve better outcomes.

For nearly all the problems tested, GI successfully refines the code generated by LLMs. The code improvements from GI typically surpass the results obtained from

Table 5.5: Median number of passed test cases (scaled in $[0, 1]$) for each problem and LLM. We highlight in bold the methods that are better for the same problem and LLM in a statistically significant way. The GI is based on $\mathcal{F}$ with lexicase selection.

	CL7			L3			CG			G4		
	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI
BS	✓	—	—	✓	—	—	✓	—	—	✓	—	—
BB	0	0	0	0	0	0.00	0.00	0.02	**0.04**	0.00	0.04	**0.04**
BW	0.00	0	**0.01**	0.00	0.00	**0.07**	0.00	0.01	**0.01**	0.03	0.42	0.11
CS	0	0	0	0	0	0	0	0	**0.28**	0	0.15	0.00
CC	0	0	**0.29**	✓	—	—	✓	—	—	✓	—	—
CV	0	**0.06**	0.00	0.17	0.30	**1**	0.92	0.49	0.99	0.85	0.85	**1.00**
DG	0.00	0	**0.01**	0.00	0.14	**0.14**	0.07	0.07	**0.14**	0.14	0.00	0.14
FP	✓	—	—	0	**0.76**	0	✓	—	—	✓	—	—
FB	✓	—	—	✓	—	—	✓	—	—	✓	—	—
FC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
GD	✓	—	—	✓	—	—	✓	—	—	✓	—	—
IS	0.78	0.78	0.78	✓	—	—	✓	—	—	✓	—	—
LD	0.22	0.22	**0.52**	0.58	1	1	✓	—	—	✓	—	—
LH	0	0	**0.04**	✓	—	—	✓	—	—	✓	—	—
MM	0.06	0.06	**0.17**	0.10	0.14	**0.72**	✓	—	—	✓	—	—
MC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
PD	0	**1**	0.17	✓	—	—	✓	—	—	✓	—	—
SL	0	0	0	0	0	0	0	0	0	0	**0.50**	0
SD	0.04	0.04	0.04	0.04	0.02	**0.07**	0.04	0.04	**0.75**	0.04	0.04	**0.08**
SB	0	0	`np`	0.00	0	**0.50**	0.00	0.62	**0.50**	0	**0.68**	0.50
SW	✓	—	—	0.35	0.67	**0.38**	✓	—	—	✓	—	—
SQ	0	0	0	✓	—	—	✓	—	—	✓	—	—
SC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
TW	✓	—	—	✓	—	—	✓	—	—	✓	—	—
VD	0	0	0	✓	—	—	✓	—	—	✓	—	—

merely re-prompting the LLMs, indicating that GI offers a significant advantage over relying solely on LLMs. This improvement is largely due to the use of a specialized grammar that steers the GI search process toward more promising areas of the search space. However, this approach depends heavily on the quality of the initial solution provided by the LLM. While the initial solution might not be entirely correct, it must offer a reasonable set of functions, libraries, and constants for GI to work with effectively. It is generally easier for an LLM to generate such a set than to produce fully correct code, which makes even smaller LLMs valuable for providing initial inputs to the GI process. Consequently, evolution tends to be more effective with smaller models and simpler problems, whereas its impact is less pronounced with more complex problems.

Regarding the evolutionary strategy, experiments reveal that the use of lexicase selection and a fitness function with finer granularity (referred to as $\mathcal{I}_{\mathsf{L}}^{\mathcal{F}_\mathcal{E}}$) enhances the quality of the solutions produced. Specifically, $\mathcal{I}_{\mathsf{L}}^{\mathcal{F}_\mathcal{E}}$ statistically outperforms $\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$ in 11 cases, while the latter surpasses the former in just one instance. This

Table 5.6: Median number of passed test cases (scaled in $[0, 1]$) for each problem and LLM. We highlight in bold the methods that are better for the same problem and LLM in a statistically significant way. The GI is based on $\mathcal{F}_{\mathcal{E}}$ with tournament selection.

	CL7			L3			CG			G4		
	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI	$\mathcal{L}$	$\mathcal{L}^+$	GI
BS	✓	—	—	✓	—	—	✓	—	—	✓	—	—
BB	0	0	0.00	0	0	**0.02**	0.00	0.02	**0.04**	0.00	0.04	**0.04**
BW	0.00	0	**0.01**	0.00	0.00	**0.01**	0.00	0.01	**0.01**	0.03	0.42	0.10
CS	0	0	0	0	0	0	0	0	**0.01**	0	0.15	**0.39**
CC	0	0	**0.29**	✓	—	—	✓	—	—	✓	—	—
CV	0	0.06	0.28	0.17	0.30	**1.00**	0.92	0.49	0.99	0.85	0.85	**0.99**
DG	0.00	0	**0.00**	0.00	0.14	**0.14**	0.07	0.07	**0.14**	0.14	0.00	0.14
FP	✓	—	—	0	**0.76**	0	✓	—	—	✓	—	—
FB	✓	—	—	✓	—	—	✓	—	—	✓	—	—
FC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
GD	✓	—	—	✓	—	—	✓	—	—	✓	—	—
IS	0.78	0.78	0.78	✓	—	—	✓	—	—	✓	—	—
LD	0.22	0.22	**0.22**	0.58	1	1	✓	—	—	✓	—	—
LH	0	0	**0.04**	✓	—	—	✓	—	—	✓	—	—
MM	0.06	0.06	**0.17**	0.10	0.14	**0.71**	✓	—	—	✓	—	—
MC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
PD	0	**1**	0.16	✓	—	—	✓	—	—	✓	—	—
SL	0	0	0.00	0	0	0	0	0	0	0	**0.50**	0
SD	0.04	0.04	0.04	0.04	0.02	**0.06**	0.04	0.04	**0.75**	0.04	0.04	**0.08**
SB	0	0	`np`	0.00	0	**0.50**	0.00	0.62	0.50	0	**0.68**	0.50
SW	✓	—	—	0.35	0.67	**0.37**	✓	—	—	✓	—	—
SQ	0	0	0	✓	—	—	✓	—	—	✓	—	—
SC	✓	—	—	✓	—	—	✓	—	—	✓	—	—
TW	✓	—	—	✓	—	—	✓	—	—	✓	—	—
VD	0	0	0	✓	—	—	✓	—	—	✓	—	—

improvement is particularly noticeable when applied to the outputs of smaller LLMs, where there is more room for correction.

Table 5.7: Median number of passed test cases (scaled in $[0, 1]$) for each problem and LLM. We highlight in bold the methods that are better for the same problem and LLM in a statistically significant way.

	CL7		L3		CG		G4	
	$\mathcal{I}^{\mathcal{F}}_{\mathsf{T}}$	$\mathcal{I}^{\mathcal{F}}_{\mathsf{L}}$	$\mathcal{I}^{\mathcal{F}}_{\mathsf{T}}$	$\mathcal{I}^{\mathcal{F}}_{\mathsf{L}}$	$\mathcal{I}^{\mathcal{F}}_{\mathsf{T}}$	$\mathcal{I}^{\mathcal{F}}_{\mathsf{L}}$	$\mathcal{I}^{\mathcal{F}}_{\mathsf{T}}$	$\mathcal{I}^{\mathcal{F}}_{\mathsf{L}}$
BB	0	0	0	0.00	0.04	0.04	0.04	0.04
BW	0.01	0.01	0.09	0.07	0.01	0.01	0.11	0.11
CS	0	0	0	0	0.00	**0.28**	0.10	0.00
CC	0.29	0.29	—	—	—	—	—	—
CV	0.00	0.00	1	1	0.99	0.99	0.99	1.00
DG	0.00	**0.01**	0.14	0.14	0.14	0.14	0.14	0.14
FP	—	—	0	0	—	—	—	—
IS	0.78	0.78	—	—	—	—	—	—
LD	0.34	**0.52**	1	1	—	—	—	—
LH	0.04	**0.04**	—	—	—	—	—	—
MM	0.17	0.17	0.71	**0.72**	—	—	—	—
PD	0.48	0.17	—	—	—	—	—	—
SL	0	0	0	0	0	0	0	0
SD	0.04	0.04	0.04	**0.07**	0.75	0.75	0.08	0.08
SB	`np`	`np`	0.50	0.50	0.50	0.50	0.50	0.50
SW	—	—	0.37	**0.38**	—	—	—	—
SQ	0	0	—	—	—	—	—	—
VD	0.00	0	—	—	—	—	—	—

Table 5.8: Median number of passed test cases (scaled in $[0, 1]$) for each problem and LLM. We highlight in bold the methods that are better for the same problem and LLM in a statistically significant way.

	CL7		L3		CG		G4	
	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}\varepsilon}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}\varepsilon}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}\varepsilon}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}}$	$\mathcal{I}_{\mathsf{T}}^{\mathcal{F}\varepsilon}$
BB	0	0.00	0	**0.02**	0.04	0.04	0.04	0.04
BW	0.01	**0.01**	**0.09**	0.01	0.01	0.01	**0.11**	0.10
CS	0	0	0	0	0.00	0.01	0.10	**0.39**
CC	0.29	0.29	—	—	—	—	—	—
CV	0.00	0.28	1	1.00	0.99	0.99	0.99	0.99
DG	0.00	0.00	0.14	0.14	0.14	0.14	0.14	0.14
FP	—	—	0	0	—	—	—	—
IS	0.78	0.78	—	—	—	—	—	—
LD	0.34	0.22	1	1	—	—	—	—
LH	0.04	**0.04**	—	—	—	—	—	—
MM	**0.17**	0.17	0.71	0.71	—	—	—	—
PD	0.48	0.16	—	—	—	—	—	—
SL	0	0.00	0	0	0	0	0	0
SD	0.04	0.04	0.04	0.06	0.75	0.75	0.08	0.08
SB	np	np	0.50	0.50	**0.50**	0.50	**0.50**	0.50
SW	—	—	0.37	0.37	—	—	—	—
SQ	0	0	—	—	—	—	—	—
VD	0.00	0	—	—	—	—	—	—

Table 5.9: Median number of passed test cases (scaled in $[0, 1]$) for each problem and LLM. We highlight in bold the methods that are better for the same problem and LLM in a statistically significant way.

	CL7		L3		CG		G4	
	$\mathcal{I}_{\mathtt{L}}^{\mathcal{F}}$	$\mathcal{I}_{\mathtt{L}}^{\mathcal{F}\varepsilon}$	$\mathcal{I}_{\mathtt{L}}^{\mathcal{F}}$	$\mathcal{I}_{\mathtt{L}}^{\mathcal{F}\varepsilon}$	$\mathcal{I}_{\mathtt{L}}^{\mathcal{F}}$	$\mathcal{I}_{\mathtt{L}}^{\mathcal{F}\varepsilon}$	$\mathcal{I}_{\mathtt{L}}^{\mathcal{F}}$	$\mathcal{I}_{\mathtt{L}}^{\mathcal{F}\varepsilon}$
BB	0	0.00	0.00	0.00	0.04	0.04	0.04	0.04
BW	0.01	**0.01**	0.07	0.09	0.01	0.03	0.11	0.10
CS	0	0	0	0	0.28	0.02	0.00	0.62
CC	0.29	0.29	—	—	—	—	—	—
CV	0.00	**0.35**	1	1	0.99	0.99	1.00	1.00
DG	0.01	0.01	0.14	0.14	0.14	0.14	0.14	0.14
FP	—	—	0	0.00	—	—	—	—
IS	0.78	0.78	—	—	—	—	—	—
LD	0.52	0.52	1	1	—	—	—	—
LH	0.04	0.04	—	—	—	—	—	—
MM	0.17	0.17	0.72	0.72	—	—	—	—
PD	0.17	0.79	—	—	—	—	—	—
SL	0	0	0	0	0	0	0	0
SD	0.04	0.04	0.07	0.09	0.75	0.75	0.08	**0.12**
SB	`np`	`np`	0.50	0.50	0.50	0.50	**0.50**	0.00
SW	—	—	**0.38**	0.38	—	—	—	—
SQ	0	0.00	—	—	—	—	—	—
VD	0	**0.00**	—	—	—	—	—	—

Chapter 6

Personalized Intepretability Estimation in Machine Learning Problems via Human-in-the-Loop Genetic Programming

Contents

In the previous chapters, we have explored a pair of important applications of Genetic Programming (GP) for addressing specific problems concerning safe stream ciphers design and automatic code generation and improvement. In this chapter, we describe a human-in-the-loop framework that, by means of GP, builds Machine Learning (ML) models in the form of trees that are accurate and interpretable according to the user that is interacting with the system. The interpretability of the models is assessed by a neural network trained using user's feedback, enabling the GP to consider human-centric criteria alongside traditional performance metrics. This bi-objective approach aims to produce models that are not only accurate but also transparent and understandable, addressing the growing demand for interpretable Artificial Intelligence (AI) [2, 3]. Moreover, while we focus our work on Symbolic Regression (SR) problems because of their popularity and importance in the scientific community [35, 36], the framework presented in this chapter is generally applicable to every machine learnable problem [49]. This work is an extension of [50].

6.1 Background

In recent years, it has been shown that ML models that are implemented and employed without adequate control and supervision may pose important risks [2]. To address this issue, the field of Explainable Artificial Intelligence (XAI) has emerged,

focusing on methods to explain model predictions and generate models that are inherently understandable by humans [186, 187]. The growing importance of XAI research is largely driven by the use of highly complex, opaque models (i.e., black-box models) in critical real-world applications where users must interact with these models to scrutinize and comprehend the reasoning behind their predictions.

There is general agreement that methods for explaining black-box models have inherent limitations (due to their underlying assumptions), and should thus be applied cautiously [188, 189]. In fact, if data permits the development of a highly accurate interpretable model, such a model should be prioritized over its black-box alternative [3]. Particularly, the field of Interpretable Machine Learning (IML) specifically focuses on the development of techniques that are able to devise inherently interpretable models that do not potentially need post-hoc explanations as in the case of black-box models such as Deep Neural Networks (DNNs).

GP is a type of Evolutionary Algorithm (EA) used to discover interpretable models such as formulae, decision trees, rule sets, or computer programs [5]. GP evolves models built from fundamental operations or instructions. If the operations used are easily understandable and GP manages to find an accurate model, there is a chance that the resulting model may also be interpretable, containing a manageable number of understandable components. However, relying solely on chance to find interpretable models is not likely to be effective. To guide GP toward discovering models that are interpretable, it is necessary to incorporate an objective function that approximates interpretability. The challenge, though, is that interpretability is inherently subjective, varying based on the specific application and the user's background [190, 1]. Additionally, different contexts may require varying balances between model accuracy and interpretability, depending on the importance of the decisions being informed by the model [191, 20].

6.2 Problem Formulation

Let X denote the input space and Y denote the output space. Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in X, \mathbf{y}_i \in Y, i \in [1..N]\}$ with $|D| = N$, where $\mathbf{x}_i$ represents the input features and $\mathbf{y}_i$ represents the corresponding labels. The dataset D is referred as training set and the elements that it contains are referred as training examples. The goal consists in discovering a model $f : X \to Y$ that is both accurate and interpretable. In particular, a model is accurate if it produces the correct label for most of the provided observations. A model is interpretable if it is considered clearly understandable by the users that must interact with it, depending on the tasks they have to perform and the information they need to collect. Building on that, we can state that the definition of interpretability is inherently subjective and, furthermore, no formal and objective definition of interpretability exists [1].

We assume that the accuracy of a model can be expressed through a function $q_f : F_{X \to Y} \to \mathbb{R}$, where $F_{X \to Y} = \{f \mid f : X \to Y\}$ represents the model space or hypothesis space. There are several well-established accuracy metrics, such as the Root Mean Squared Error (RMSE) for regression tasks (i.e., when $Y = \mathbb{R}$), and classification accuracy for classification problems (i.e., when Y is a finite set with no inherent ordering).

When it comes to interpretability, we assume there exists a measure $\hat{\psi}_f : F_{X \to Y} \to$

$\mathbb{R}$, which reflects the user's subjective view of what makes a model interpretable.

We focus on the scenario where the user aims to tackle a supervised learning problem by providing a dataset D and selecting the preferred accuracy measure q_f, but they may not be able to specify their interpretability measure $\hat{\psi}_f$ (e.g., because the user finds it difficult to formalize). However, we assume the user is available to provide feedback on the interpretability of models produced during the learning process.

6.3 Proposed Method

The problem can be addressed by using a bi-objective optimization algorithm where the search space is $F_{X \to Y}$ and the objective function is composed by the accuracy and the interpretability, with both of them that must be maximized. While the accuracy can be calculated by using an exact and deterministic objective measure, the interpretability $\hat{\psi}_f$ needs to be estimated, given the complexity or impossibility to have an exact definition. Specifically, an approximation of the interpretability is calculated by using an estimator ψ_f that is concurrently trained by using user's feedback during the learning process itself. For the sake of simplicity, we are going to refer to the solutions evolved by the optimization process as *models* and we are going to use the term *estimator* only for ψ_f, even though the evolved ML models are estimators themselves for other tasks.

The aforementioned approach is defined as Model Learning with Personalized Interpretability Estimation (ML-PIE) [50, 49].

6.3.1 Learning Process

The learning process consists of an iterative optimization process that searches for good models according to the provided objectives and an active learning strategy that iteratively collects examples that are adopted to train the estimator. The training procedure of the interpretability estimator is what makes this framework be labeled as human-in-the-loop. As a matter of fact, the active learning algorithm selects pairs of models that are evolved during the optimization process. These pairs of models are presented to the user that is prompted to provide a feedback on their interpretability. This system concurrently gathers the user's feedback during the model search process in an asynchronous manner: the rate at which models are created is not tied to the rate at which the user reviews and provides feedback on a subset of them.

The rationale behind using a concurrent approach, rather than an off-line one where user's feedback is gathered before the model search begins, is that the user interacts with the models as they are being discovered. This ensures that ψ_f is trained on data that is in-distribution for the model search process, reflecting the actual distribution of models found. In contrast, an off-line method offers no such assurance that user's feedback would correspond to the distribution of models being uncovered by the search algorithm.

At initialization step, ML-PIE begins with a randomly generated estimator ψ_f. Alternatively, ψ_f can be set to an estimator that has been pre-trained, for example, using previously annotated data from one or more users [51], or based on an interpretability proxy from existing literature (e.g., [309]). Several options for initializing

ψ_f are discussed in Sec. 6.3.3.

At each iteration, the model search algorithm generates a set of candidate models, evaluating them using q_f and the current ψ_f. ML-PIE then adds the syntactically unique models discovered by the model search algorithm to an initially empty set $\tilde{F} \subset F_{X \to Y}$, which is reset at each iteration to include only models relevant to the current search status. Simultaneously, the active learning algorithm selects models that are adopted to refine ψ_f iteratively.

In each iteration, the active learning algorithm formulates a query by selecting a pair of candidate models f_1 and f_2 from $\tilde{F}$ based on a priority rule (described in Sec. 6.3.3), and presents this pair to the user for evaluation. The user is tasked with selecting the model they find more interpretable through a graphical interface. Upon receiving the user's input, the active learning algorithm updates ψ_f using this feedback as a signal.

For instance, if the user considers f_1 more interpretable than f_2, then ψ_f will be updated to either increase the difference $\psi_f(f_1) - \psi_f(f_2)$ if $\psi_f(f_1) > \psi_f(f_2)$ (i.e., positive reward), or decrease the difference $\psi_f(f_2) - \psi_f(f_1)$ otherwise (i.e., negative reward). Once ψ_f has been adjusted, the active learning algorithm formulates a new query, initiating the next iteration of the user's feedback collection process.

It is worth noting that feedback collection is framed as a simple binary choice between two models to ease the user's cognitive load. This setup is flexible across various model spaces $F_{X \to Y}$. Asking the user to rank more than two models would likely increase the complexity of the task, while requesting them to directly score a single model (e.g., on a scale of one to ten) might also prove difficult. Additionally, designing an appropriate scoring mechanism for the user can be challenging for system designers.

The feedback collection process, i.e., the active learning algorithm, continues to operate in parallel with the model search algorithm until the latter concludes. The model search ends when a predetermined budget, such as a maximum number of iterations (or generations, in GP), or a time limit, is exhausted.

In conclusion, ML-PIE runs two concurrent iterative processes. The model search algorithm updates one or more models per iteration, driven by a fixed q_f and a continuously evolving ψ_f, proceeding as fast as the computational environment allows. The active learning algorithm, on the other hand, updates ψ_f at a pace determined by the rate at which the user provides feedback.

6.3.2 Applicability

ML-PIE is a general approach that can be applied to many and diverse scenarios and use cases, provided that:

(a) Models in $F_{X \to Y}$ can be processed by the estimator ψ_f that is updated during the learning process. This requirement can be satisfied if models can be encoded in a numerical representation that captures relevant model properties (a concrete case is discussed in Sec. 6.3.3);

(b) Given a pair of models, it must be feasible for a user to tell which one is the most interpretable in the pair at hand. This requirement can be satisfied if models can be represented by using a user-friendly visual representation, so

that user's feedback can be collected via, for instance, Graphical User Interface (GUI).

Regarding the model search algorithm, which operates in $F_{X \to Y}$, we impose a bi-objective framework, though this requirement can be relaxed if necessary by employing techniques like linearization or lexicographic prioritization of objectives.

Lastly, a practical consideration is that the speed of the model search algorithm must roughly align with the user's feedback rate. In other words, the algorithm should allow enough time for the user to provide a sufficient amount of feedback (we explore this aspect in more detail in Sec. 6.4.4).

6.3.3 Applying ML-PIE to Symbolic Regression

ML-PIE can be applied to tackle SR problems, which are considered to be NP-Hard [37]. SR is a type of regression task that aims to find a mathematical expression $f : \mathbb{R}^d \to \mathbb{R}$, in symbolic form, that best fits a given set of data points $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in \mathbb{R}^d, \mathbf{y}_i \in \mathbb{R}, i \in [1..N]\}$ with $|D| = N$, where $\mathbf{x}_i$ represents the input features and $\mathbf{y}_i$ represents the corresponding target. Unlike traditional regression, which fits predefined functions (e.g., linear, polynomial), SR searches for the optimal mathematical model without specifying in advance the form of the function. The objective is to discover both the structure of the model or formula and the values of its coefficients.

Ideally, f must be realized by composition of basic and interpretable functional building blocks. These building blocks are selected by the user and are an input for the system. In this context, $X = \mathbb{R}^d$, $Y = \mathbb{R}$, and the space of models $F_{X \to Y}$ is the space $\mathcal{S}_d$ of formulae for d variables. Consequently, the interpretability estimator $\psi_f : \mathcal{S}_d \to \mathbb{R}$ takes a formula as input.

We employ GP as the model search algorithm, given its demonstrated effectiveness in SR tasks [35, 36]. GP is configured to operate in a bi-objective framework, simultaneously optimizing for both accuracy and interpretability. Upon completion, GP produces a set of formulae that reflect varying trade-offs between these two objectives, allowing the user to select the model that best suits their requirements. Further details on the bi-objective GP approach are provided in Sec. 6.3.3.

Regarding the active learning algorithm for refining the interpretability estimator ψ_f, we explore various design alternatives. Each approach follows a similar framework. First, we use an encoding function $h : \mathcal{S}_d \to \mathbb{R}^m$ to transform a model (i.e., a formula) into a numerical vector, where the m components correspond to predefined features of the formula. These features need to be chosen in advance, as detailed in Sec. 6.3.3.

Next, we use an Artificial Neural Network (ANN) to estimate the interpretability score $\psi_f(f)$ for a formula f, by feeding the feature vector of f as input to the ANN, i.e., $\psi_f(f) := \text{ANN}_{\boldsymbol{\theta}}(h(f))$, where $\boldsymbol{\theta} \in \mathbb{R}^p$ are the parameters of the network. Over time, by using an online learning approach, these parameters are adjusted to align the ANN predictions with the user's feedback, assuming the encoding function h captures relevant and informative features. For more details concerning how neural networks work, see Appendix C.

We experiment with various options for the encoding and for optimizing the ANN.

In addition to updating ψ_f through optimization of $\boldsymbol{\theta}$, the active learning algorithm is tasked with generating queries, i.e., selecting which pairs of formulae to present to the user for evaluation.

Practically, user feedback is gathered via a graphical interface (shown in Fig. 6.1), where ML-PIE displays two formulae, and the user selects which one they find more interpretable with a single click.

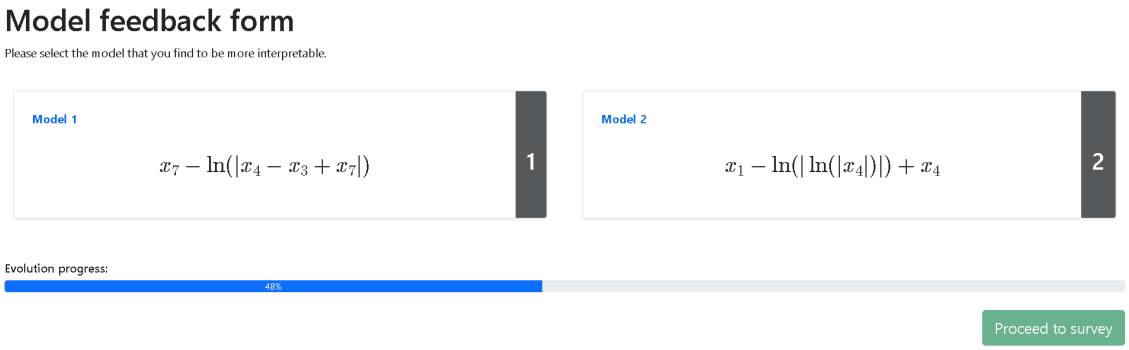

Figure 6.1: A screenshot of the user interface during the feedback collection process.

Model Search

We employ the NSGA-II algorithm [310] as a bi-objective model search method, using the implementation provided in [292]. More specifically, we leverage a GP-based version of NSGA-II, encoding the formulae as trees [311]. NSGA-II is chosen due to its popularity in multi-objective evolutionary optimization settings. However, we stress that the choice of the optimization algorithm is not the core aspect of this work; rather, the key component lies in how we compute the interpretability estimate. In fact, this system could also be implemented by replacing NSGA-II with another multi-objective optimization algorithm. We believe NSGA-II is the most convenient choice considering the scope of this work, given its widespread use and demonstrated speed and effectiveness for multi-objective discrete optimization tasks.

The population is initialized by using ramped half-and-half algorithm [312, 5, 313] with a maximum depth of four. We avoided generating deep trees since they can easily hinder interpretability. In each generation, a new set of offspring is produced from the current population. The process begins with selecting promising parents using a tournament selection strategy with a tournament size of two. Offspring are then generated in a quantity equal to the population size through sub-tree crossover and mutation. Specifically, 90 % of the offspring are created by crossover between randomly chosen pairs of parents (each pair yielding two offspring), while the remaining 10 % are clones of randomly selected parents. Afterward, 60 % of the offspring, chosen randomly, undergo sub-tree mutation. Offspring whose depth exceeds $l_{\max} = 4$ (as excessively large formulae tend to lack interpretability) or that duplicate members of the current population or previously generated offspring are discarded and regenerated. Duplicates are avoided as maintaining population diversity is crucial for enhancing NSGA-II performance in the context of GP [314, 315]. Two trees are considered identical if they are semantically equivalent (e.g., $x_1 + x_2$ is the same as $x_2 + x_1$). This is determined by comparing their outputs, so if the predictions are identical, the trees are treated as the same, even if their syntax differs.

Once the offspring are generated, NSGA-II performs a final selection step by combining the current population from the start of the generation with the newly created offspring. The selection is based on non-domination sorting and crowding distance to ensure diversity and convergence. This process also incorporates an elitist strategy, following NSGA-II core mechanism, which helps retain the best individuals. The result of this selection forms the population for the next generation.

We adopt the following building blocks as nodes for the evolved trees:

- The variables $x_1, \ldots, x_d$ of the problem, i.e., the features of the dataset;
- Ephemeral random constants uniformly sampled from $[-5, 5]$;
- The mathematical operators $+$, $-$, $\times$, $\div^*$, $(\cdot)^3$, $\ln^*$, max (where * denotes the protected version[1] to avoid operations performed on out-of-domain values).

We configured the population size to be 200 and ran the algorithm for a total of 50 generations. These parameters were selected after conducting multiple preliminary tests, as they strike a balance between algorithm performance and execution duration. Specifically, the aim is to allow the algorithm enough time to uncover accurate models, while maintaining a manageable runtime for the user. With this setup, a single GP run takes roughly 10 minutes in a machine with Ubuntu 20.04, 64 GB of RAM, and a AMD EPYC 7542 32-core processor.

As previously discussed, NSGA-II seeks to optimize both accuracy and interpretability simultaneously. For accuracy, we maximize the R^2 score, which serves as the coefficient of determination, thus focusing on improving model accuracy. To refine the fit of the generated formulae, we incorporate linear scaling [316, 317], a technique known for its effectiveness on real-world datasets [318].

For interpretability, we utilize the predictions from the estimator ψ_f, which aims to reflect the user's perception of how interpretable a given formula is. Since the estimator ψ_f evolves with user feedback, the interpretability of the formulae is re-evaluated at each generation. Ultimately, NSGA-II produces a set $S^\star \subset \mathcal{S}_d$ of formulae that represents a Pareto-optimal balance between accuracy and interpretability, with each formula offering a different trade-off between the two goals.

Model Encoding

We explore three different approaches for implementing the encoding function $h : \mathcal{S}_d \to \mathbb{R}^m$, each varying in size, complexity, and expressiveness. Within the context of SR, where formulae are depicted as trees, the function h is designed to operate on these tree structures. It is worth noting that this tree representation can be adapted to various other applications, such as regular expressions or decision trees, highlighting the versatility of our method.

The first approach is named *counts encoding*. In this approach, a tree is represented as a vector where each element corresponds to a specific primitive, with the value of each element indicating the frequency of that primitive within the tree. Specifically, $|\mathcal{F}|$ elements are dedicated to tracking the occurrences of various mathematical operators, where $\mathcal{F} = \{+, -, \times, \div^*, (\cdot)^3, \ln^*, \max\}$. Additionally, d elements are utilized to count the occurrences of each feature relevant to the problem, while one element accounts for the number of numerical constants present in the tree.

[1] $\div^*(a, b) = \text{sign}(b)\ \frac{a}{|b| \text{ if } |b| > 10^{-9} \text{ else } 10^{-9}}$; $\ln^*(a) = \ln(|a| + 10^{-9})$

Moreover, the encoding vector contains three additional components that convey important information regarding the tree's structure and size: (i) the ratio of the number of depth levels to the total number of nodes; (ii) the ratio of the maximum number of operands an operator in the tree can accommodate to the maximum breadth (i.e., the greatest number of nodes at the same depth level); (iii) the percentage of leaf nodes.

Consequently, the total size of this encoding is given by $m = |\mathcal{F}| + d + 1 + 3$. This encoding is beneficial because it scales with $\mathcal{F}$ and d, meaning it is not affected by the maximum number of nodes in the tree. However, it does not differentiate between primitives based on their positions within the tree, which could influence interpretability.

The second approach is named *one-hot encoding*. Unlike the previous encoding, the one-hot encoding method considers the precise position of each element within the tree. Specifically, every node is represented as a one-hot encoded vector of size $|\mathcal{F}| + d + 1$, where all components are initialized to zero except for the one corresponding to the element in that node (with the last $+1$ representing the constants), which is set to one. The encoding $h(f)$ is created by concatenating the encodings of all nodes as the tree is traversed in a breadth-first manner.

To maintain consistency in encoding, we assume that all nodes possess the maximum possible number of operands (maximum arity α): if a node is absent, its corresponding encoding will be $\mathbf{0} \in \mathbb{R}^{|\mathcal{F}|+d+1}$.

The one-hot encoding is evidently more expressive than the counts encoding since it can represent a broader range of tree properties that are relevant to interpretability. However, it does not scale as efficiently compared to the counts encoding because it is dependent on $|\mathcal{F}|$, d, and the maximum potential number of nodes in a tree of maximum depth $l_{\max}$. In particular, the size of the encoding is given by $m = \frac{\alpha^{l_{\max}}-1}{\alpha-1}(|\mathcal{F}| + d + 1)$.

The third approach is named *level-wise counts encoding*. This final encoding strikes a balance between the counts encoding and the one-hot encoding methods. It records the frequency of operators, problem features, and constants at each depth level $l \leq l_{\max}$ within the tree. Additionally, similar to the counts encoding, it includes three descriptive components related to the shape and size of the tree.

The overall size of the level-wise counts encoding is given by $m = l_{\max}(|\mathcal{F}| + d + 1) + 3$, which ensures it remains relatively compact in relation to the maximum number of levels $l_{\max}$, the size of the function set $|\mathcal{F}|$, and the number of problem features d. However, unlike the one-hot encoding, this approach does not retain information regarding the specific positions of nodes.

Fig. 6.2 shows the scaling of the size m of the three encodings for different combinations of $|\mathcal{F}|$, d, α, and $l_{\max}$.

Estimator Optimization

We employ an ANN as the estimator for interpretability. The ANN takes the encoded formula as input and outputs an interpretability score, expressed as $\psi_f(f) = \text{ANN}_{\boldsymbol{\theta}}(h(f))$. To assess prediction uncertainty, we utilize dropout [319] during the prediction phase [320], which facilitates uncertainty-based active learning. While there are alternative methods for obtaining uncertainty measures from an ANN,

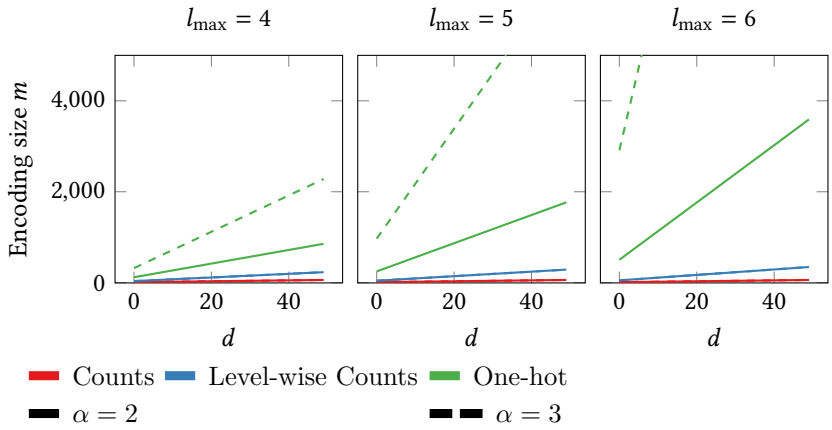

Figure 6.2: Encoding sizes w.r.t. the amount of problem features d, the maximum tree depth $l_{\max}$, and the maximum node arity α. The size of the function set $\mathcal{F}$ is 7, as in Sec. 6.3.3.

such as bootstrapping or conformal prediction [321], we opted for dropout due to its seamless integration within our framework.

In terms of the ANN architecture, we utilize two hidden layers, each comprising 100 neurons activated by ReLU functions [322]. The output layer contains a single node with a tanh activation function. Additionally, we incorporate dropout in both hidden layers [319] with a probability set at 0.25. This dropout mechanism is applied during both the training and evaluation phases.

During training, the neural network processes the encoded formula in a single forward pass to generate the signal required for optimizing the ANN parameters (details provided below). Conversely, for inference, we execute $k = 10$ predictions (forward passes) for the same formula to accommodate the randomness introduced by dropout. We then compute the mean of these k predictions to derive the interpretability estimate, while the standard deviation serves as the measure of uncertainty (refer to Sec. 6.3.3).

To optimize the parameters $\boldsymbol{\theta}$ of the ANN, we utilize a binary signal derived from a comparison between two formulae, reflecting the specific type of feedback we seek from the user (which formula is preferred). Specifically, a training sample for adjusting $\boldsymbol{\theta}$ consists of a triplet comprising two formulae, f_1 and f_2, along with a binary label $p \in \{-1, 1\}$. If the user considers f_1 to be more interpretable than f_2, we assign $p = -1$; otherwise, $p = 1$. Using this information, we calculate the interpretability scores $\psi_f(f_1)$ and $\psi_f(f_2)$ (performing one forward pass for each), and then we compare these scores to the label p. We then employ the Wasserstein loss function [323] for training the ANN, defined as follows:

$$\mathcal{L}\left(\psi_f\right) = p\left(\psi_f\left(f_1\right) - \psi_f\left(f_2\right)\right), \tag{6.1}$$

which has been applied in a similar context in [50].

The loss will yield a positive value if there is a discrepancy between the user's preferences and the estimator assessment regarding the relative interpretability of f_1 and f_2; conversely, it will be negative if they align. It is important to note that this loss function is designed to train a ranking ANN, rather than a conventional regressor ANN. In essence, the ANN does not learn to predict scores that possess an

inherent meaning; instead, these scores are only significant in a comparative context, serving to rank various models. As demonstrated by [50], this training approach is effective and necessitates less annotation effort compared to the estimation of explicit scores (i.e., a regression method). Furthermore, the model search algorithm utilized, NSGA-II, primarily relies on rankings rather than absolute values. The sole exception is the computation of the crowding distance, which is normalized at each generation using the minimum and maximum values for each objective encountered.

We employ Adam as optimizer [324], with a learning rate of 10^{-3} and a weight decay of 10^{-5}.

We contend that employing an ANN as the interpretability estimator for an online learning approach is a logical decision. This is due to the fact that, by utilizing a specific loss function, we can incrementally refine the model of the interpretability estimator (i.e., the weights of the ANN). This is feasible because we can compute gradients incrementally as user feedback is collected. In particular, our feedback mechanism involves determining which of two provided formulae is preferable. To effectively capture this feedback, we utilize a Wasserstein-like loss function. Furthermore, we aim to incorporate a method for assessing prediction uncertainty. If we were to adopt an alternative model, such as a decision tree, the process of training the estimator based on user feedback would become a binary classification problem, which would eliminate our ability to utilize incremental gradients and compute uncertainty.

As outlined in Sec. 6.3.1, the ML-PIE framework begins its search for a formula f by employing an initial interpretability estimator ψ_f^0. In practice, this initial estimator can significantly influence the models that are ultimately identified, as it guides the model search algorithm along a specific trajectory.

Consequently, we explore the possibility of incorporating an optional warm-up phase before the formal initiation of ML-PIE. Specifically, this involves supplying ML-PIE with an initial estimator ψ_f^0 that is derived from an optimization conducted on a dataset of triplets $\{(f_1^{(i)}, f_2^{(i)}, p^{(i)})\}_i$. This dataset is pre-collected and independent of the current problem, but it remains compatible with it in terms of the sets $\mathcal{F}$ and d. We will discuss the methods for obtaining the warm-up signal p in more detail below. It is important that the warm-up phase is not excessively brief (regarding the number of training triplets provided) as that would be ineffective, nor should it be overly long, as this may risk premature convergence to a sub-optimal interpretation of interpretability that may not align with the user's perspective. Our preliminary experiments indicate that utilizing around 20 triplets yields satisfactory results.

ϕ-driven warm up. We utilize the interpretability estimator ϕ, as proposed by [51], to assign labels to 20 pairs of randomly generated formulae, which are based on the function set $\mathcal{F}$ and d problem variables. For each random pair of formulae f_1 and f_2, we assign $p = -1$ if $\phi(f_1) \leq \phi(f_2)$, otherwise we set $p = 1$. The estimator ϕ evaluates formulae based on several factors, including the number of components, the number of operations, the presence of non-arithmetic operations, and the length of consecutive non-arithmetic operation sequences. We note that the encodings introduced in Sec. 6.3.3 are designed to be more general than those used by ϕ, which is specifically tailored to SR.

Data-driven warm up. Another possible and more general strategy involves exploring a data-driven warm-up phase. In this scenario, we aim to utilize a dataset containing well-structured and elegant models relevant to the problem domain, such as those sourced from publicly available databases. For SR, we specifically select a subset from the 100 equations presented in the Feynman Lectures on Physics [325, 36]. We construct 20 triplets f_1, f_2, p by taking f_1 directly from this dataset and generating f_2 by applying random mutations to f_1, leading to an increase in the number of nodes within the corresponding tree. The assumption made is that each f_2, by virtue of its increased complexity, is less interpretable than its paired f_1, and the label p is assigned accordingly. To maintain balance within the dataset, for 50 % of the triplets, we swap f_1 with f_2 and invert the corresponding label p. We chose to employ a small sample of data to better align with real cases in which the user is not likely to provide a large number of test cases for the execution of an eventual warm-up phase.

It is worth noting that this data-driven method is not limited to SR but can be adapted to other types of models as well. For instance, this technique could be applied to the automatic generation of regular expressions by leveraging a collection of examples written by humans. Notably, regular expressions can be expressed in tree form and evolved using GP [326].

Query Construction

The user is shown a pair of formulae $(f_1, f_2) \in \hat{F}$, where $\hat{F}$ represents the set of distinct formulae present in the current generation's population. These formulae, f_1 and f_2, are selected by the active learning mechanism. We explore two different strategies for implementing active learning.

We employ random sampling without replacement, meaning that random formulae f_1 and f_2 are selected from $\hat{F}$, ensuring that the user does not encounter the same formula more than once. While this method is straightforward and quick, it has a limitation: it does not actively seek to maximize the information gained for training the ANN. Nevertheless, random sampling can still prove useful, particularly when dealing with highly dimensional input spaces (i.e., when the encoding is of considerable complexity).

The second approach leverages uncertainty sampling (based on the ANN predictions), a commonly used technique in active learning, which was shown to be effective in [50]. To implement this, we estimate the uncertainty of each formula using the ANN with dropout enabled during prediction, as described in Sec. 6.3.3. Specifically, for each formula in the set $\hat{F}$ (the current generation's formulae in NSGA-II), the ANN generates $k = 10$ predictions, each differing due to the dropout mechanism. The uncertainty for a formula is calculated as the standard deviation of these predictions. Finally, we select the two formulae from $\hat{F}$ with the highest uncertainty.

It is evident that uncertainty sampling demands more computational effort than random sampling, since it necessitates calculating the uncertainty for each formula in $\hat{F}$ with every query. This evaluation cannot be pre-computed because the uncertainty values may change between queries. This is due to the updates made to ψ_f, which alter the uncertainty associated with the same formula over time.

6.4 Experimental Analysis

We organize the experimental phase into three primary sections, each with specific objectives:

(1) Evaluate the capability of the estimator ψ_f to be trained in alignment with user preferences, independent of model search; (2) Examine how varying user profiles (feedback rates) influence the results of the model search in GP; (3) Implement the most effective configuration of ML-PIE identified in the previous experiments with real users to evaluate its performance.

In the first experiment, we explore various methods to simulate the user's actual perception of interpretability by using different Proxies of Human Interpretability (PHIs). This is done across different encoding strategies h, warm-up approaches, and active learning sampling methods. For the second experiment, we execute ML-PIE with simulated users who behave according to these PHIs, employing distinct response patterns. Finally, in the third experiment, we involve 42 participants, consisting of B.Sc., M.Sc., and Ph.D. students from engineering, computer science, and AI courses, to provide feedback on two real-world datasets.

All experimental phases are guided by the understanding that the proposed system is designed for real human users. We assume that participants will not be highly responsive and may be unwilling to give extensive feedback. Realistically, it is unlikely that a user will spend hours offering feedback to train the interpretability estimator. Despite this, we aim to evaluate the system's performance even when the feedback is limited. Therefore, the experiments with the estimator are carried out with a small number of training pairs to reflect a realistic scenario. Similarly, the GP experiments are run with a reduced number of generations and smaller population sizes to ensure that the optimization process remains brief, minimizing the risk of user fatigue or disinterest.

Our implementation is developed using Python 3.9.12 and can be accessed at the following repository: `https://github.com/lurovi/ML-PIE`. For data pre-processing and training of the neural networks, we utilize Scikit-learn [327] and PyTorch [328], respectively. The GP evolution is handled through GenePro [329] and PyMOO [292]. It is worth noting that, unless explicitly stated otherwise, all parameters are set according to the values originally used by [50] in the ML-PIE framework.

6.4.1 Proxies of human interpretability

For the first and second experiments, we rely on simulated users to explore and fine-tune important settings of ML-PIE before involving real participants in the third experiment. These simulated users provide feedback based on a predefined PHI. This approach enables us to gather extensive feedback in a consistent and controlled way. The PHIs under consideration are as follows:

- ℓ-PHI: This metric evaluates formulae purely by their size, meaning the number of nodes in their tree representation. In this case, smaller formulae are deemed more interpretable, without regard to the specific primitives they contain;
- ϕ-PHI: This PHI follows the interpretability metric ϕ outlined in [51], which factors in the formula size, the number of operations, the count of non-

arithmetic operations, and the number of consecutive applications of non-arithmetic operations;

- W-PHI: This method assigns distinct weights to each primitive (including functions from $\mathcal{F}$, problem features, and constants). The interpretability of a formula is then determined by summing the weights of the primitives present in its tree;
- LW-PHI: An extension of W-PHI, this level-wise PHI assigns weights to primitives based on their depth in the tree. The weight varies according to the depth level where the primitive appears;
- NW-PHI: A further extension of W-PHI and LW-PHI, this PHI assigns weights to primitives considering both their type and their specific position in the tree. It is the most comprehensive PHI model we use.

For W-PHI, LW-PHI, and NW-PHI, we simulate different user profiles by sampling the weights with

$$w = -1 \times |\rho|, \text{ with } \rho \sim \mathcal{N}\left(0, \sigma^2\right).$$

The multiplication by -1 reflects that the inclusion of each component reduces the interpretability of the formula, while σ^2 is adjusted specifically for each type of primitive. We assign smaller values of σ^2 to simple operations (such as $+, -, \times, \div$), as well as to variables and constants, since these elements are typically easy to comprehend. Conversely, larger values are allocated to more complex operators (e.g., ln and max), as these are generally perceived to be more challenging to understand.

It is important to observe that, by design, our PHIs are inherently negative. Additionally, we acknowledge that the PHIs we propose do not necessarily reflect any objective measure of interpretability. For instance, relying on model size has been a point of contention in interpretability research. Instead, these PHIs serve as stand-ins to emulate potential user preferences.

6.4.2 Datasets

We conduct our GP-based experiments on two datasets for supervised regression tasks:

- **Boston housing** (Boston): a dataset used to derive models that predict house prices across different neighborhoods in Boston [330]. This dataset is commonly used in AI fairness studies, as it includes a feature related to race. It consists of 506 samples and 13 features;
- **Heating load** (Heating): a dataset used for modeling the prediction of heating load requirements in buildings, containing 768 instances and 8 features [331, 332].

For both datasets, we perform three different random 7:3 splits between training and test sets. The features are normalized by using robust scaling [327]. Each split undergoes 10 separate GP runs.

6.4.3 Interpretability Estimator Training

We train the ANN with up to 150 training pairs, selected based on both active learning strategies, from a training set of 500 randomly generated trees. We select 150 feedback iterations as it approximates the maximum amount of feedback that we expect users to provide. Each pair is labeled according to a specific PHI, which serves as a proxy for a user's interpretability preferences. After every feedback round, we calculate the Spearman footrule [333, 334], a metric that quantifies the disparity between two rankings, on a validation set of 300 randomly generated trees. The random formulae generated for this phase use six variables, $x_1, \ldots, x_6$. Each experiment is repeated 10 times with different random seeds, and we explore all combinations of the following parameters: (1) Encoding h (Sec. 6.3.3); (2) PHI (Sec. 6.4.1); (3) Active learning method (Sec. 6.3.3); (4) Warm-up strategy (Sec. 6.3.3).

Fig. 6.3 presents the findings from this experiment. The results are summarized as follows:

- **Encoding**: The counts encoding generally performs the best across different PHIs and active learning approaches, though the differences are not statistically significant. This encoding, characterized by its low dimensionality and scalability (see Sec. 6.3.3), is effective for learning simple PHIs like ℓ-PHI, but struggles with more complex PHIs such as NW-PHI (as indicated by the footrule convergence). In contrast, the one-hot encoding, which has the largest dimensionality, tends to over-fit on simple PHIs like ℓ-PHI (particularly with random sampling, as shown by the solid line). However, it outperforms counts encoding on more complex PHIs like NW-PHI, though this requires a large volume of feedback. Given the limited feedback in human-in-the-loop scenarios, smaller encodings appear to be better suited compared to larger ones;
- **PHI**: As anticipated, ℓ-PHI is the easiest for the estimator to learn, regardless of encoding, due to its simplicity. Nonetheless, we observe that more complex ground truths, such as ϕ-PHI and NW-PHI, lead to decreasing footrule values over time. This implies that a more general encoding, with sufficient feedback, enables the estimator to approximate complex PHIs;
- **Active learning**: The results indicate that random sampling generally performs as well as, if not better than, uncertainty-based sampling. Only in specific scenarios, such as W-PHI with Feynman warm-up and counts encoding, uncertainty-based sampling offers a modest advantage over random sampling. We discuss this further later in the section;
- **Warm-up**: In most configurations, particularly with counts encoding, the use of warm-up contributes to a lower footrule score when no feedback is provided. However, after about 20-25 rounds of feedback, the gap between using warm-up and not using it closes. Additionally, warm-up does not typically lead to premature convergence to a sub-optimal estimator. In general, there is no significant difference between the two warm-up strategies we explored.

To further elucidate some of our earlier findings, we carry out additional experiments where we assess both the average uncertainty and the footrule on the validation set, with the estimator trained using a significantly larger amount of feedback. Fig. 6.4 illustrates the performance of the ANN trained with up to 1000 training pairs. For

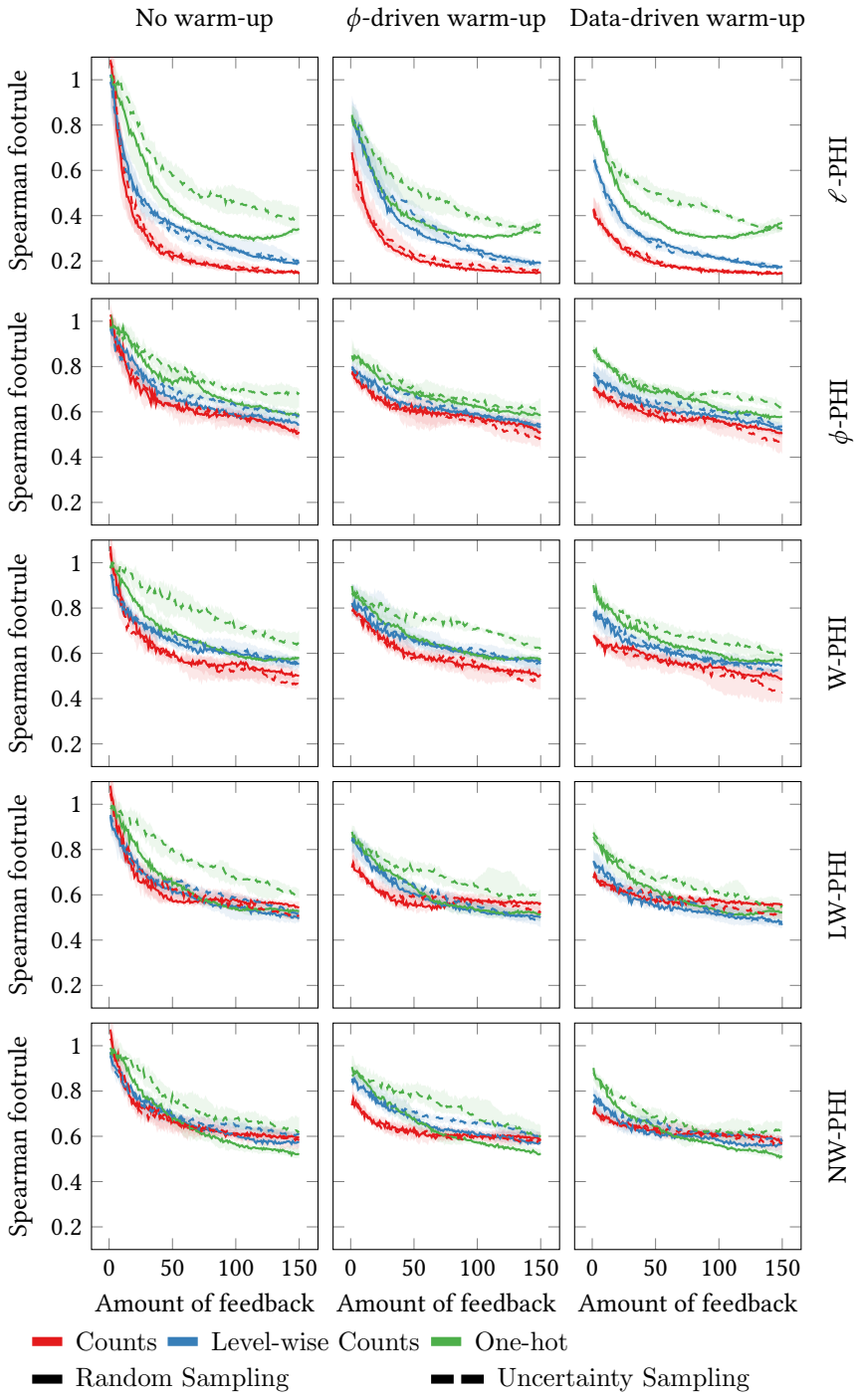

Figure 6.3: Median and inter-quartile range of the Spearman footrule with increasing amount of feedback for different formulae encodings, sampling methods, warm-up procedures, and different PHIs.

this experiment, we use ϕ-PHI to label the feedback, and we exclude any warm-up phase. Here, uncertainty sampling consistently outperforms random sampling, on average, after receiving 200–400 feedback points, regardless of the encoding used. This result confirms our hypothesis that uncertainty-based active learning can, indeed, surpass random sampling. However, depending on the complexity of the encoding, a substantial amount of feedback may be necessary before this advantage becomes apparent.

Based on the aforementioned insights, we opt to use counts encoding, random sam-

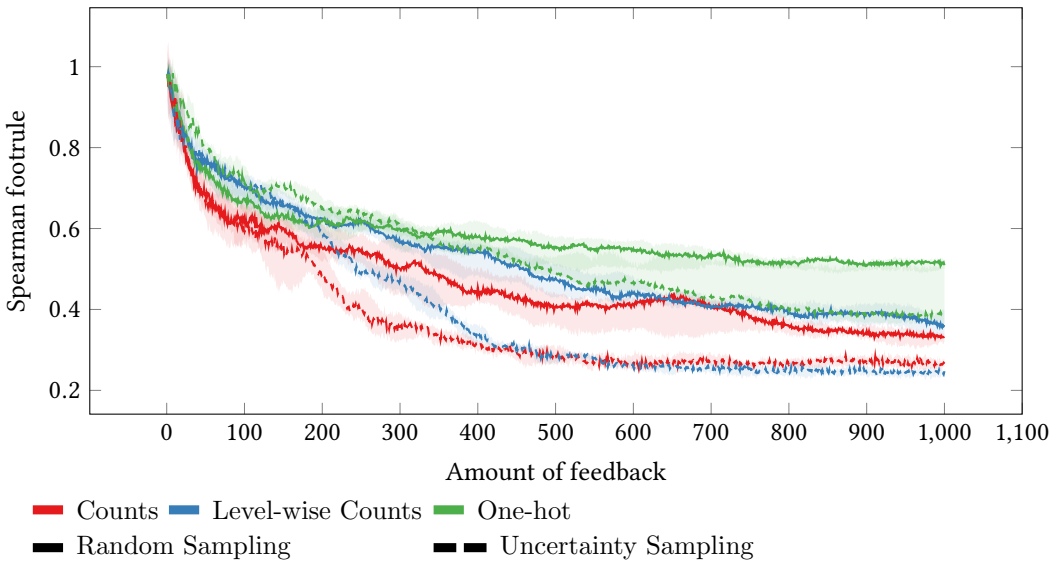

Figure 6.4: Median and inter-quartile range of the spearman footrule with increasing amount of feedback for different formulae encodings and sampling strategies with no warm-up and ϕ-PHI.

pling, and the ϕ-based warm-up for the subsequent experiments. The rationale behind selecting the latter is supported by statistical tests: Tab. 6.1 presents whether using a warm-up strategy is significantly better than no warm-up, particularly when only a small number of five feedback interactions is available. We choose the ϕ-based warm-up because it consistently achieves lower p-values compared to the data-driven warm-up across various PHIs (except for ℓ, which is notably simplistic), making it preferable in cases where the user may provide minimal feedback.

Table 6.1: p-values retrieved from Bonferroni-corrected [308] Wilcoxon paired tests [293] of the distribution of footrules, across 10 runs, at five feedback rounds. We use random sampling in the active learning strategy.

PHI	ϕ-driven vs. no warm-up	Data-driven vs. no warm-up
ℓ	0.097	0.004
ϕ	0.004	0.008
W	0.008	0.008
LW	0.004	0.027
NW	0.004	0.020

6.4.4 GP with Simulated Users

In this section, we perform an in-depth analysis of the complete ML-PIE process, where multi-objective GP operates in tandem with user feedback (in this case, simulated) to train the interpretability estimator. As noted at the conclusion of the previous section, and driven by the findings from that analysis, we employ the following configuration: counts encoding, random sampling, and ϕ warm-up.

In particular, we aim to address the following key questions:

1. How do varying user profiles impact the accuracy of the discovered models?

Specifically, does the user's personal perception of interpretability significantly affect GP's search process?

2. How resilient is the proposed approach to differing levels of user engagement and feedback frequency?

To investigate these questions, we conduct a two-part experimental evaluation, detailed below.

Trade-off between Accuracy and Interpretability

We configure ML-PIE with a simulated user who provides feedback at regular intervals of 5 s, guided by one of the selected PHIs. The chosen PHIs include a subset of the most reasonable options: ℓ-PHI, ϕ-PHI, and NW-PHI. We incorporate ℓ-PHI and ϕ-PHI as they represent commonly accepted notions of interpretability [51], while NW-PHI is included as it captures the most general concept among those introduced. The model search is repeated 10 times for each combination of dataset, train-test split, and PHI, resulting in a total of $10 \cdot 2 \cdot 3 \cdot 3 = 180$ runs. At the conclusion of each run, the solutions are re-evaluated on the test set to estimate their generalizability.

For each Pareto front obtained from the training set, we select formulae corresponding to varying degrees of interpretability and evaluate their accuracy. To achieve this, we rank the solutions within each front based on their estimated interpretability, and denote their rank position using the percentile τ. A value of $\tau = 100$ represents the highest interpretability but lowest accuracy, while $\tau = 0$ indicates the opposite, i.e., minimal interpretability and maximal accuracy. This approach enables us to compare models across different interpretability measures (PHIs).

Before presenting the quantitative results, Tab. 6.2 provides examples of models along with their accuracy for various interpretability percentiles τ, generated by GP. Qualitatively, we observe that as τ increases, the formulae become simpler but also less accurate, which aligns with expectations.

Fig. 6.5 presents the distribution of train and test accuracy—measured by the R^2 score—for the models mentioned earlier, across $\tau \in \{0, 25, 50, 75, 100\}$. Above each triplet of box-plots (representing the three PHIs), we provide the p-values derived from the Kruskal-Wallis statistical test [307], which tests the null hypothesis that the distributions are not significantly different. A $\star$ indicates instances where the p-value is less than 0.01.

Fig. 6.5 illustrates the relationship between interpretability and accuracy. Specifically, across all PHIs and datasets, a higher value of τ corresponds to a decrease in the R^2 scores. This trend is anticipated and acts as a sanity check to confirm the proper functioning of ML-PIE. Assuming each PHI reflects a distinct user profile or subjective view of interpretability, we can analyze the distributions of R^2 based on the PHIs employed for each percentile τ and evaluate whether differing PHIs result in variations in model accuracy. Notably, the reported p-values indicate that in 8 out of 10 instances for the training set, and in 7 out of 10 instances for the test set, the samples appear to be drawn from different distributions. Therefore, in the majority of cases, employing a different PHI leads, indeed, to variations in accuracy levels. This suggests that the trade-off is significantly influenced by the user's subjective interpretation of interpretability. Consequently, these findings im-

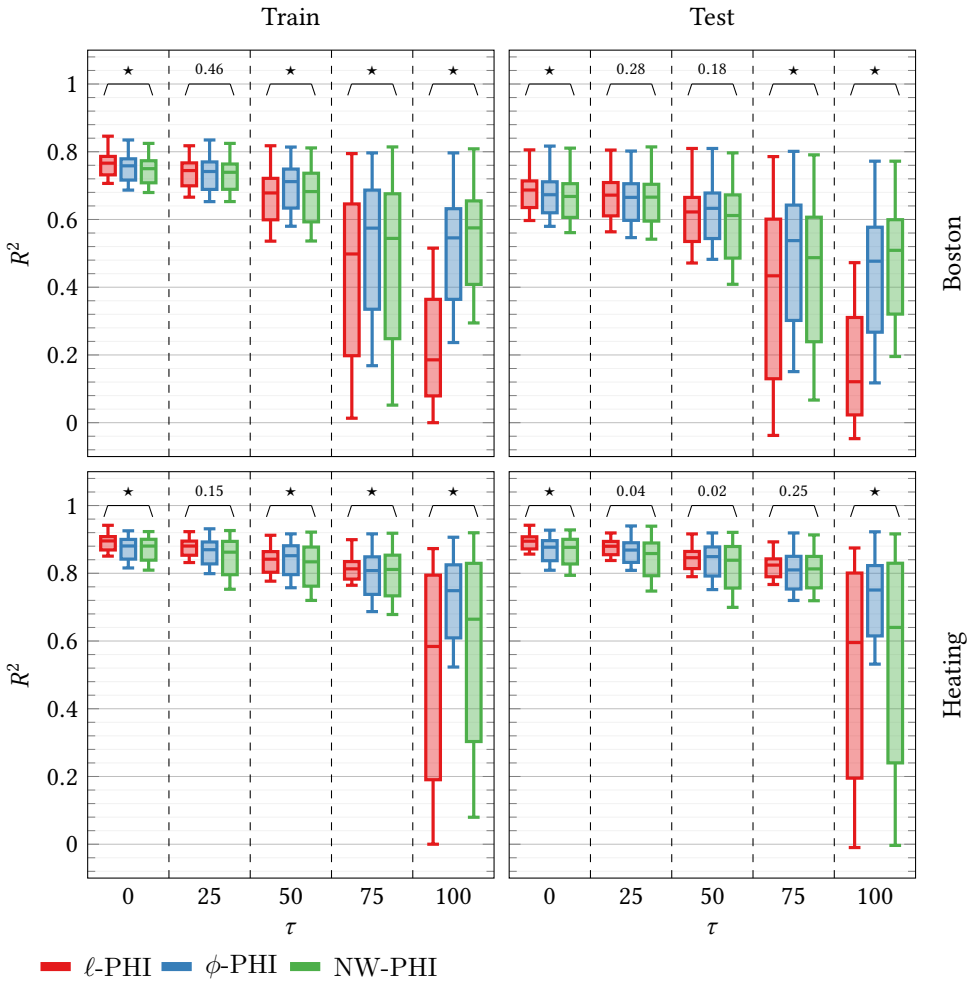

Figure 6.5: Distribution of the R^2 for different τ on both training and test sets w.r.t. the different simulated user profiles. For each τ, statistical significance is reported.

ply that GP may require different configurations for different users. For instance, if a user prioritizes model accuracy but their interpretability criteria complicate the identification of accurate models, it may be necessary for GP to operate with a larger budget.

Search Algorithm Robustness

In the preceding experiment, we modeled a constant user response rate. In this section, we aim to determine whether varying this response rate affects the behavior of GP. Therefore, we replicate the previous experiment using two distinct user engagement profiles: a lazy-start profile and a lazy-end profile. For the lazy-start profile, the simulated user provides feedback every 8 s during the first half of the model search (measured in generations) and every 2 s for the remainder of the experiment. Conversely, in the lazy-end profile, feedback is given every 2 s in the first half and every 8 s in the second half. Additionally, we introduce a further configuration where we utilize the estimator trained with a constant response rate of 5 s from the beginning of the evolution, effectively treating it as an oracle. In this scenario, the user does not participate, meaning the oracle remains unchanged. Similar to the previous experiment, we conduct 10 iterations for each configuration, resulting in a total of $10 \cdot 2 \cdot 3 \cdot 3 \cdot 3 = 540$ runs, and at the conclusion of each run, we reassess all solutions on the test set.

In the prior experiment, our focus was on determining whether varying PHIs results in different levels of accuracy. In this section, we aim to investigate whether the overall Pareto fronts differ when the user's response rate is altered. To accomplish this, we utilize the Hyper-Volume (HV) indicator, which quantifies the size of the area covered by the solutions within the Pareto front [335]. Generally, a greater HV signifies superior search performance, reflecting a larger section of the search space that is dominated by the solutions. Given that we train ANNs for ranking rather than for predicting specific interpretability scores, the ANNs yield distinct scores across different runs, even when trained on the same PHI. Consequently, these scores lead to HVs that are not directly comparable. To address this, we calculate the PHI for each formula when computing the HV. This approach is justified, as the estimator is designed to rank models similarly to how the PHI operates. Furthermore, to facilitate comparisons of HVs across varying PHIs, we apply min-max normalization to the PHI scores, ensuring that all values are scaled to the interval $[0, 1]$.

Fig. 6.6 illustrates the distribution of HVs across different user profiles, organized by dataset (one row per dataset), PHI (one column per PHI), and train/test set (on the x-axis). Additionally, we present the p-values derived from a Kruskal-Wallis statistical test [307] above each set of four box plots, where the null hypothesis posits that the distributions are equal.

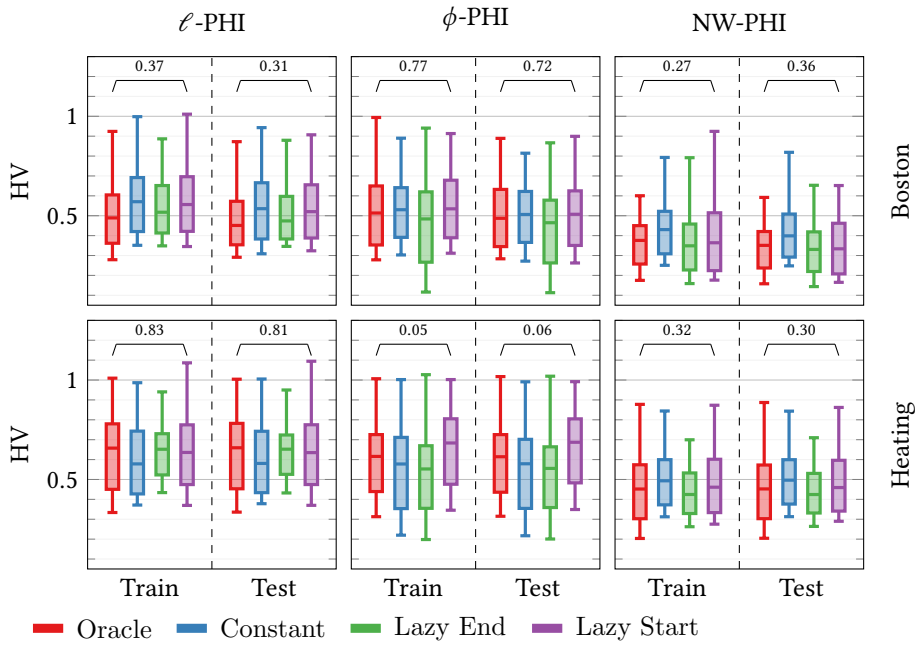

Figure 6.6: HVs distribution for different engagement profiles.

The plots and corresponding p-values indicate that there are no significant differences in the HVs resulting from varying engagement profiles. Thus, we can deduce that ML-PIE demonstrates resilience against different user behaviors, as long as sufficient feedback is provided during the model search process. Furthermore, based on the outcomes observed with the oracles, we find that the model search remains unaffected by the changing objectives of interpretability over time.

These findings bolster the hypothesis that an approach like ML-PIE is worth pursuing. In fact, involving a human in the loop may ultimately enhance efficiency for the user, as they can annotate relevant models identified during the model search,

Table 6.2: Examples of models found by the bi-objective GP with a simulated user (NW-PHI), employing random sampling within active learning and a ϕ-driven warm-up for the interpretability estimator. For each dataset, we show the results of three runs. We sample models according to their interpretability percentile (τ) in the Pareto front. Higher τ means higher interpretability. We report the R^2 score of each model on both train and test sets.

		R^2		
Dataset	τ	Train	Test	Model
Boston	10	0.81	0.59	$\max(x_{12}, -0.15 - 4.39) - \max(x_9, x_5) + x_9 - x_3$
		0.77	0.78	$x_{12} - x_3 - x_{12} - \max(x_{12}, x_5)$
		0.75	0.72	$x_{12} - \max\left({x_1}^3 x_3 x_6, x_5 - x_{12} + x_{10}\right)$
	50	0.71	0.56	$x_{12} - x_5 - x_3 + x_{10}$
		0.72	0.68	$x_{12} - \max(x_{12} - \max(x_6, x_5), x_5)$
		0.72	0.63	$(x_4 + x_{10})\, x_5 - x_5 + x_{12}$
	90	0.57	0.54	$x_5 - x_8$
		0.27	0.29	$x_{11} - x_{11} + x_5 - x_8 - x_0$
		0.42	0.43	$x_{12} + \max(x_7, x_2 - x_7 - x_7)$
Heating	10	0.9	0.9	$x_6 - \ln(\lvert x_0 - \ln(\lvert x_4 \rvert)\rvert)$
		0.9	0.86	$x_6 + 4.61 x_2 + 2.33^3 (x_6 - x_3) - \max(x_2 x_2, x_5 + x_0) - \ln\left(\lvert \frac{x_4}{-3.75} \rvert\right)$
		0.93	0.94	$\ln(\lvert 0.14 - x_0 \rvert) - x_2 - \left(x_6 - x_0 - -2.22^3\right)\left({x_6}^3 + x_4\right)$
	50	0.82	0.79	$x_0 - x_4 - x_0 - \ln(\lvert x_3 \rvert)$
		0.9	0.86	$x_1 - x_4 - x_2 - x_4 - x_6$
		0.91	0.93	$x_3 - 4.5 - x_4 - x_6 + x_4 - x_0 - x_4$
	90	0.7	0.7	$x_6 - x_1 - x_4$
		0.8	0.73	$x_6 - x_3 - x_0 - x_2 - x_6$
		0.68	0.66	$x_0 - x_4 - x_0 - x_6 - x_0$

rather than relying on a collection of examples gathered offline, which could be out-of-distribution for a particular instance of model search.

6.4.5 GP with Real Users

In our final experiment, we implement ML-PIE with real users. We engage a total of 42 participants, consisting of B.Sc., M.Sc., and Ph.D. students from the computer science and engineering departments at the University of Trieste, Italy. No rewards are provided to the participants for their involvement in the study.

Each participant is introduced to a web interface that outlines the task: to repeatedly select which of two formulae they find more interpretable. This task is conducted once for the Boston dataset and once for the Heating dataset. The order in which the datasets are presented is randomized for each participant.

Upon clicking the "start on dataset Boston/Heating" button, the ML-PIE process begins, and the interface shown in Fig. 6.1 is displayed. ML-PIE operates under the same parameters as in prior experiments, utilizing a population size of 200 and running for 50 generations, which amounts to roughly 10 minutes of runtime per dataset in our computational resources.

Once ML-PIE completes its processing on a dataset, users are directed to a subsequent feedback page. On this page, they encounter 4 pairs of formulae. The first pair consists of the formula at $\tau = 25$ from the Pareto front generated by the specific ML-PIE run that the user has been providing feedback for, paired with another formula sourced from an offline run that utilized ϕ as its interpretability metric, selected for its accuracy closest to the first formula (for the same dataset and train-test split).

The remaining pairs are constructed in a similar fashion, adjusting τ to 5 and/or substituting ϕ with ℓ. Users are not informed which formula originated from ML-

PIE, and the order of the formulae in each pair is randomized to maintain a blind comparison.

In this evaluation, users can indicate their preference for either the first or second formula, or express that they find both formulae equally (un)interpretable. Ultimately, this survey aims to determine whether users favor the personalized approach that ML-PIE is designed to offer.

In Fig. 6.7, we present the findings from the user survey. It appears that, on average, participants prefer models generated through offline methods using either ℓ or ϕ, rather than those produced by ML-PIE. This outcome is somewhat discouraging and may stem from various factors: participants are not domain experts and were also not heavily engaged in the feedback process, as no incentives were provided.

To investigate this further, we examine the quantitative data. In Fig. 6.8, it is evident that we barely achieve 50 feedback rounds during these experiments. When we compare this with Fig. 6.3, we find that the estimator typically struggles to effectively rank models when trained with fewer than 50 feedback rounds across different PHIs. Furthermore, Fig. 6.8 indicates that the rate of mispredictions—measured by comparing the relative ranking of the two formulae as predicted by the estimator with the feedback from users—is quite substantial (0.5 indicates a random guess).

To gain insight into this outcome, we examine the findings reported in [50]. In that research, participants provided significantly more feedback on average—100 compared to the 50 by the conclusion of an ML-PIE run in our study. This indicates a higher level of engagement from users in [50].

Another significant distinction between our study and that of [50] is the encoding method utilized. Their study employed a smaller encoding scheme based on four formula features, which was better suited for SR, whereas we utilized counts encoding. Consequently, while exploring more general encodings is a promising approach for enhancing the adaptability and ease of deploying ML-PIE across various applications, it appears that such broader encodings may be too complex to be effective in a human-in-the-loop context.

Nonetheless, there is potential for improvement in developing more effective warm-up strategies that require minimal feedback to achieve satisfactory results.

A possible future analysis should be focused on involving domain experts on a specific task, since the tested scenario involved generic students from scientific areas with no prior knowledge on the application domain and whose confidence level was not assessed. Therefore, it is likely that many choices given by the participants were almost random, not reflecting an actual interpretability notion that can be learned. Driving a survey on a specific topic with actual domain experts should enable the discovery of more reliable results, where real interpretability proxies are observed.

6.5 Discussion

Based on the outcome of our experimental tests, we try to provide some general findings and observations that, despite the high customization of ML-PIE, may help in identifying which are the ideal implementation strategies of the components of this system depending on the context in which it has to be adopted.

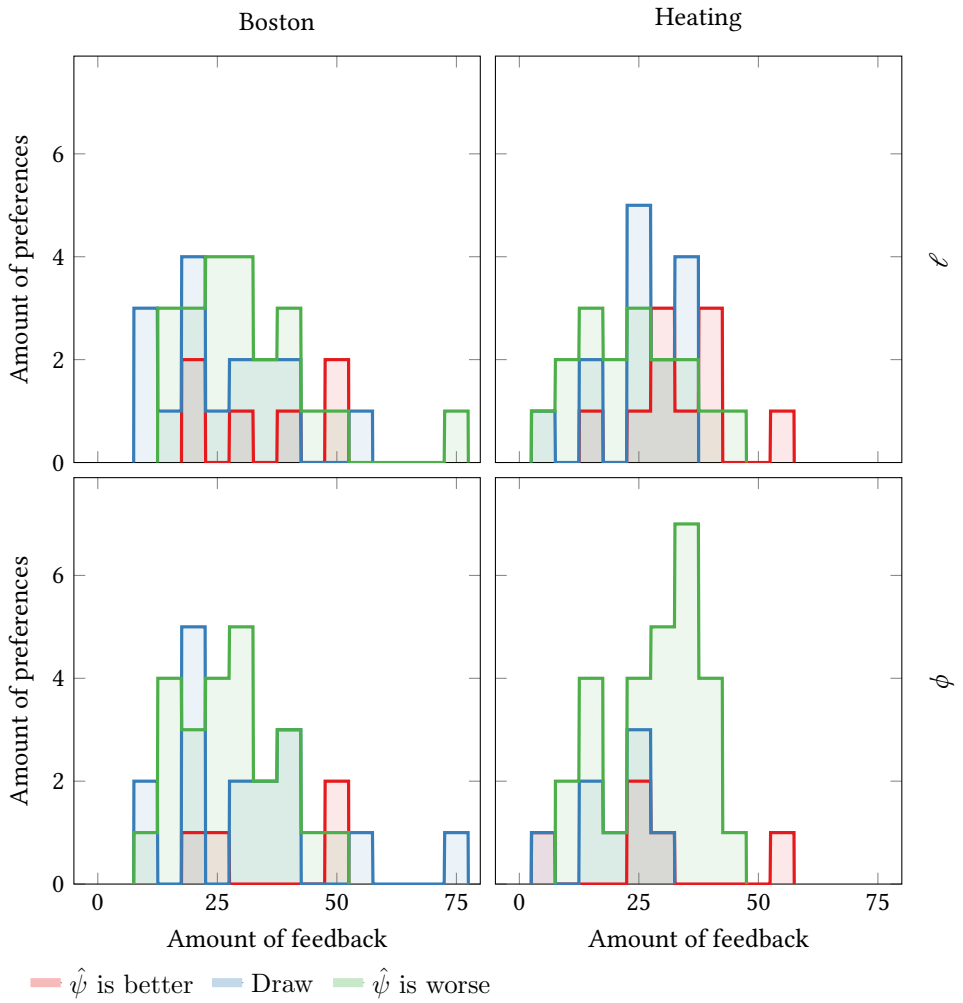

Figure 6.7: Amount of preferences for each model collected in the survey at the end of the study with real users w.r.t. the amount of feedback provided by users during the evolutionary runs. For the x-axis, we consider bins of size 5.

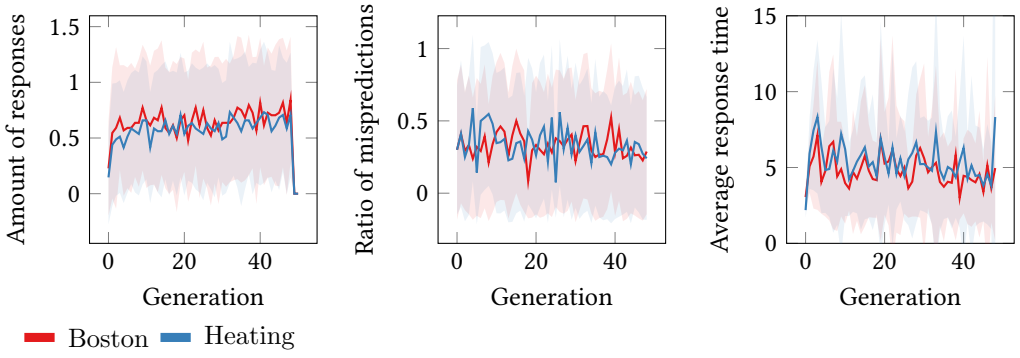

Figure 6.8: For each problem, we show the mean and standard deviation range of the amount of responses, the ratio of mispredictions the estimator makes in ranking the models (prior to the user's feedback signal), and the average response time, across the generations.

First of all, we introduce several novel and general approaches for encoding a tree structure. Each encoding method comes with its own trade-offs in terms of expressiveness and computational and space complexity. Nonetheless, our experiments show that a numerical representation of a GP tree can effectively capture a broad

range of interpretability concepts. Therefore, the selection of a specific encoding method largely depends on the available feedback and the interpretability proxy being approximated. In particular, a simple encoding scheme offers efficiency and can swiftly learn an interpretability definition, though its estimates may be imprecise if the definition is particularly complex. Conversely, a more complex encoding method, while computationally intensive, has the potential to learn any interpretability definition, regardless of its intricacy. However, even for simpler interpretability proxies, a complex encoding approach demands a substantial amount of user feedback.

In our second experiment, we simulate user intervention during the feedback gathering process with various PHIs. Different user behavior patterns, exhibiting distinct engagement trends, are also tested. From the analysis of the resulting Pareto fronts, we observe that as the formulae become more interpretable, their accuracy diminishes. This observation supports the fact that our general estimator can effectively capture the tested PHIs and be incorporated into any GP-based evolutionary process. Additionally, we find that the choice of PHI can influence the level of accuracy, meaning that the qualitative performance of the discovered models depends on the specific interpretability concept being learned. Thus, based on the interpretability notion, it may be necessary to adjust the GP parameters (e.g., population size and number of generations) accordingly. Furthermore, by analyzing various user profiles, we demonstrate that our framework remains resilient to different user behaviors, as long as a sufficient amount of feedback is provided.

Finally, we evaluate our framework using real users who participate in the feedback collection process. We emphasize that when only a small amount of feedback is provided, non-expert users in our experiments tend to favor models other than those created by ML-PIE. In contrast, models generated through ML-PIE are preferred when a substantial amount of feedback is supplied. This demonstrates the system potential for discovering interpretable models. However, depending on the specific interpretability definition being approximated, the system may require a sufficiently large quantity of feedback to achieve accurate estimations.

Chapter 7

Cellular Geometric Semantic Genetic Programming

Contents

In this final work, we aim at investigating how to enhance a specific version of Genetic Programming (GP) [5] called Geometric Semantic Genetic Programming (GSGP) [52] to solve Symbolic Regression (SR) problems. We introduce a novel approach to GSGP, defined as Cellular Geometric Semantic Genetic Programming (cGSGP), where a structure inspired by the concept of Cellular Automata (CA) is imposed over the population [253, 254, 255]. This approach is designed to enhance the evolutionary dynamics of GSGP, ensuring a more diverse and robust exploration of the solution space by limiting premature convergence [336].

7.1 Background

GP explores the search space by using operators that merely act on the syntactic representations of individuals. GSGP considers the space of the semantic representation of programs and each syntactic manipulation aims at obtaining a specific effect on the semantic, which is the behavior of a program during its execution. In our scenario, individual semantic is the vector containing the predictions on the training set [52] and thus we can map each individual into a vector on the semantic space, i.e., $\mathbb{R}^N$.

GSGP employs the Geometric Semantic Operators (GSOs) for crossover and mutation. Crossover is defined as:

$$T_{\text{XO}} = R \cdot T_1 + (1 - R) \cdot T_2 \tag{7.1}$$

where $T_1, T_2 : \mathbb{R}^d \to \mathbb{R}$ are the parents, and R is a random tree representing a real function with outputs in $[0, 1]$. For each $\mathbf{x} \in \mathbb{R}^d$, $T_{\text{XO}}(\mathbf{x})$ is a linear combination of

the two parents' outputs. This way, the offspring has semantics belonging to the segment joining the semantics of the two parents on the semantic space.

Mutation is defined as:

$$T_{\mathrm{M}} = T + m \cdot (R_1 - R_2) \tag{7.2}$$

where $T : \mathbb{R}^d \to \mathbb{R}$ is the tree being mutated, R_1, R_2 are two random trees representing real functions with outputs in $[0, 1]$, and m is the mutation step. Mutation is a weak perturbation of the individual since the new individual is contained in the hyper-sphere of radius m centered in the semantics of the old individual.

GSOs induce a unimodal fitness landscape, but also increase the size of the individuals. Crossover increases the size of the individuals exponentially in the number of generations [52].

In our work, we combine GSGP with a CA-inspired approach. CA is a formal model where identical cells, or automata, are distributed according to a given topology. Each cell updates its internal state based on a local rule that depends exclusively upon a set of neighboring cells. In our context, all individuals of a population are positioned into a toroidal grid, which is a n-dimensional torus. Hence, each individual belongs to a specific cell of the grid, which is chosen randomly at initialization time.

We consider an individual to be in the neighborhood of radius r of a given cell if it is contained in the hypercube with side $2r + 1$ corresponding to the Moore neighborhood of radius r. The n-dimensional toroidal grid with neighborhood of radius $r \in \mathbb{N}$ will be denoted by $\mathcal{T}_r^n$. We impose a toroidal configuration to the grid to have a complete neighborhood for every cell. There are no disconnected neighborhoods in $\mathcal{T}_r^n$ when $r > 0$. This means that there are no isolated or independent sub-populations.

Let i be a position in a n-dimensional toroidal grid whose neighborhood is $\mathcal{N}_i$. For instance, in two dimensions we have that $i = (i_1, i_2)$ and a neighborhood of radius r is $\mathcal{N}_i = \{j = (j_1, j_2) \mid ||i - j||_\infty \leq r\}$, where $||x||_\infty$ is the maximum norm $\max_{1 \leq i \leq n} |x_i|$. Each individual is contained in its own neighborhood.

Given a population, each position i is occupied by a tree T_i. The tree that will occupy position i in the next generation depends upon the trees contained in the positions given by $\mathcal{N}_i$.

Fig. 7.1 shows an example of two-dimensional grid with 100 individuals.

7.2 Problem Formulation

SR is a type of regression task that aims to find a mathematical expression $f : \mathbb{R}^d \to \mathbb{R}$, in symbolic form, that best fits a given set of data points $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in \mathbb{R}^d, \mathbf{y}_i \in \mathbb{R}, i \in [1..N]\}$ with $|D| = N$, where $\mathbf{x}_i$ represents the input features and $\mathbf{y}_i$ represents the corresponding target. Unlike traditional regression, which fits predefined functions (e.g., linear, polynomial), SR searches for the optimal mathematical model without specifying in advance the form of the function. The objective is to discover both the structure of the model or formula and the values of its coefficients.

GSGP, a variant of GP, is an Evolutionary Algorithm (EA) whose goal consists in addressing SR problems. Despite its benefits over traditional GP, GSGP still

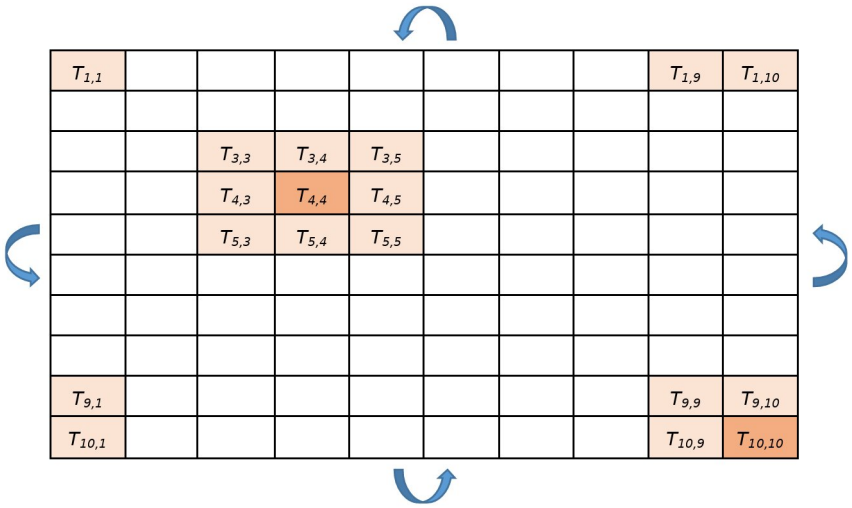

Figure 7.1: Example of a population with 100 individuals distributed according to $\mathcal{T}_1^2$. In the figure, two positions with the corresponding neighborhood are highlighted. Each cell contains the individual in that position, whose row index and column index are indicated as a subscript.

faces several challenges [70]. Specifically, although GSGP creates a unimodal fitness landscape, the fact that not all formulae within this landscape are accessible leads to the formation of local optima, where evolution can stagnate. Another issue is that this technique may encourage early convergence toward individuals that outperform others in the initial stages of evolution. This results in a shorter takeover time, which is the number of generations required for a single individual to dominate the population. Consequently, after a given amount of generations the fitness landscape can be explored only via mutation, since the convex hull of the population, which is the only area of the space that can be explored via crossover, has shrunk too much and far from the global optimum.

The goal of this work is developing a variant of GSGP that attempts to mitigate the aforementioned issues regarding the premature convergence of standard GSGP to address SR problems more effectively.

7.3 Proposed Method

The proposed method consists in implementing constraints that detail which individuals may interact between themselves during the evolution by devising a local neighborhood-based selection strategy.

The individuals in the population are organized by employing an n-dimensional toroidal grid with a neighborhood radius r, as described in Sec. 7.1. At the beginning of the evolution, the individuals are randomly generated and, then, randomly located in the various cells of the toroidal grid to perform the initialization of the method. Our main contribution consists in defining two selection strategies that take advantage of the toroidal grid to limit the propagation of dominant individuals in the early generations.

The first strategy consists in using a rank-based selection method within local neighborhoods (method defined as RKS), where all the individuals in the neighborhood

$\mathcal{N}_i$ are considered. The selection sorts the individuals of $\mathcal{N}_i$ based on the fitness, with the best individual in the first position. If crossover is not performed, position i will be filled with the best individual in $\mathcal{N}_i$. If crossover is performed, position i will be filled with the individual obtained by the crossover between the two best individuals in $\mathcal{N}_i$. Regardless of the fact that crossover is performed or not, the new individual in position i may undergo mutation, whose resulting individual will be trivially placed in the same position.

RKS is a deterministic approach that, if deprived of the constraints regarding the local interactions, would force premature convergence in a low number of generations. Even though different neighborhoods can have different best individuals, RKS may still plug an excessive amount of competition into the selection phase. Provided that, our second strategy consists in using a tournament-based selection method within local neighborhoods (method defined as TRS_p). The tournament size is determined by a real value $p \in (0, 1]$ that represents a proportion of the given neighborhood.

TRS_p is identical to RKS except for the fact that the sorting step is only applied on a subset $S \subseteq \mathcal{N}_i$. The subset S is sampled by iterating across the individuals of $\mathcal{N}_i$ and inserting them into S with probability p. Therefore, p is the parameter that enables to tune the selection pressure. By setting $p = 1$, we obtain RKS.

Following, we provide a pseudo-code for TRS_p (Algorithm 2):

Algorithm 2 Selection strategy adopted within cGSGP (`CellularSelectionGSGP`)

Input
  P population at the beginning of the current generation
  p probability of inserting an individual into a tournament
  r neighborhood radius
  n toroidal grid dimension
Output
  P_S set of pairs containing the selected parents for each position i
$P_S \leftarrow \{\}$
for $i \leftarrow 1$ to $|P|$ **do**
  $S \leftarrow \{\}$
    ▷ Extract the neighborhood with radius r of the i^{th} individual in P
  $\mathcal{N}_i \leftarrow$ `Neighborhood`(P, i, r, n)
  for $j \leftarrow 1$ to $|\mathcal{N}_i|$ **do**
    ▷ Sample a random real number in $[0, 1)$
    if `RandomUniform`$(0, 1) < p$ **then**
      $S \leftarrow S \cup \{\mathcal{N}_i[j]\}$
    end if
  end for
    ▷ Take the two individuals in S with the best fitness
  $S_{\text{BestFirst}},\ S_{\text{BestSecond}} \leftarrow$ `SelectTwoBest`(S)
  $P_S \leftarrow P_S \cup \{(S_{\text{BestFirst}},\ S_{\text{BestSecond}})\}$
end for
return P_S

Crossover and mutation are defined as the standard GSOs employed in GSGP and described in Sec. 7.1.

7.4 Experimental Analysis

We test both GSGP and cGSGP. Particularly, for cGSGP, we test radius from one to three by using a two-dimensional toroidal grid with both RKS and $\mathsf{TRS}_{0.6}$. Preliminary experiments showed that no statistically relevant difference is recorded for different values of p that are close to 0.6. We chose this value since this means sampling a subset of the entire neighborhood that is slightly larger than half of it. Such sample should provide a good and meaningful approximation of the semantic of the neighborhood without using all its individuals at the same time. Moreover, we chose to focus our analysis on two-dimensional toroidal grids since they are more commonly used in the context of CAs [260, 258].

We run our experiments on eight regression datasets that have already been used in GP-related problems [56, 65, 62, 33]. In Tab. 7.1 we show the number of observations and the number of variables for each dataset.

Table 7.1: The number of observations and the number of variables of tested datasets.

Dataset	# Observations	# Variables
Vladislavleva14 (V14) [309]	6024	5
Keijzer6 (KJ6) [316, 337]	170	1
Airfoil (ARF) [338]	1502	5
Concrete (CNC) [54]	1029	8
Slump (SLM) [339]	102	9
Toxicity (TXC) [340]	234	626
Yacht (YCH) [341]	307	6
Parkinson (PRK) [55]	5875	18

Each dataset has been partitioned in 100 train-test splits. In particular, 70 % of the observations, selected at random with uniform probability, has been used as training set, while the remaining 30 % has been used as test set. For each method and dataset we perform 100 repetitions with different seeds, one for each split[1].

We set the optimization algorithm with a single-objective minimization problem that adopts as fitness function the Root Mean Squared Error (RMSE) [21] computed between the target and the predictions obtained by applying the discovered symbolic formulae on the training set.

Trees are initialized by using the ramped half-and-half algorithm [312, 5, 313] with a maximum depth of 6 (root has depth 0). Crossover probability is set to 0.90. Mutation probability is set to 0.60. The mutation step is sampled uniformly at random from $[0, 1]$ at each mutation event. The function set is composed by $+$, $-$, $*$, $\div$ (protected) and the terminal set is composed by the variables of the problem along with 100 constants sampled uniformly at random from $[-100, 100]$. Elitism with a single elite individual is utilized.

[1] KJ6 is defined for a single split and thus we use the same split for all repetitions, with the seed influencing only the evolution.

We test different pairs of population size and number of generations as detailed in Tab. 7.2. We choose the population sizes to obtain three different perfectly squared two-dimensional toroidal grids. We set the number of generations to obtain the same number of fitness evaluations and perform a fairer comparison[2].

Table 7.2: Combinations of population size and number of generations that are tested. For each combination, the side length of the squared toroidal grid is shown, along with the corresponding total number of fitness evaluations.

Population size	Generations	Side length	Fitness evaluations
100	1000	10	100 000
400	250	20	100 000
900	111	30	99 900

We run each technique by using: crossover and mutation together sequentially ($\mathcal{E}_{\text{cx+mut}}$), only crossover ($\mathcal{E}_{\text{cx}}$), and only mutation ($\mathcal{E}_{\text{mut}}$). From now on, these variants fall under the name of *exploration pipeline*.

Statistical significance is assessed by performing a Kruskal-Wallis test [307] with $\alpha = 0.05$ for the involved methods and the given dataset and, in case the methods are meaningfully different, using a Wilcoxon signed-rank test [293], where p-values are collected and adjusted by using a Holm-Bonferroni correction [308] with $\alpha = 0.05$.

To ensure that the observed results are attributable to the communication topology in cGSGP, we conduct tests on GSGP using tournament selection with varying tournament sizes, up to a maximum of 12. When statistically analyzing the results, no significant differences is detected. Therefore, we report the findings by comparing cGSGP with GSGP using a tournament size of 4. The code for the experiments is available in the following repository: https://github.com/lurovi/CA-GSGP.

7.4.1 Statistical Comparison

The results of our experiments are shown in the following tables (Tabs. 7.3 to 7.5), in which the median RMSE values on the test sets are presented. Within the same table, for each dataset, all topologies, selection strategies, and variants of exploration pipeline are presented.

For a given dataset, cGSGP methods that are statistically significantly better than the methods in the table with the same selection strategy and exploration pipeline, including the corresponding GSGP, are marked with an asterisk (*). We highlight in bold the cGSGP methods that, taken individually, are statistically significantly better than the GSGP method with the same exploration pipeline.

In general, for $\mathcal{E}_{\text{cx+mut}}$, across all datasets, cGSGP consistently outperforms GSGP, with no substantial difference observed between the specific selection methods (RKS or $\text{TRS}_{0.6}$). The only outlier appears to be the KJ6 dataset when the population size is set to 100.

When the population size is small, the results are largely influenced by the topology: a minimum neighborhood size is required. For instance, $\mathcal{T}_1^2$, which has the smallest

[2]Since the total number of fitness evaluations is not a multiple of 900, we set the corresponding number of generations to the nearest rounded integer available.

Table 7.3: Table of the median best fitness for cGSGP on the test set and all considered topologies.

	100 as population size				1000 generations			
Method	V14	KJ6	ARF	CNC	SLM	TXC	YCH	PRK
				$\mathcal{E}_{\text{cx+mut}}$				
GSGP	1.47	1.72	2.6	6.5	4.38	2262.13	3.69	8.35
$\mathcal{T}_1^2$ w/RKS	**1.23**	1.88	2.53	6.47	4.29	**2274.38**	3.88	8.4
$\mathcal{T}_2^2$ w/RKS	**1.47**	3.08	**2.36**	**6.13**	**4.11**	**2194.0**	**3.32**	**8.16**
$\mathcal{T}_3^2$ w/RKS	**1.47**	3.84	**2.34***	**5.89***	4.13	**2174.52***	**3.22***	**8.05***
$\mathcal{T}_1^2$ w/$\text{TRS}_{0.6}$	1.06	1.75	2.64	6.61	4.5	2317.46	4.09	8.5
$\mathcal{T}_2^2$ w/$\text{TRS}_{0.6}$	1.31	2.48	**2.43**	**6.29**	**4.05**	**2243.8**	**3.44**	**8.22**
$\mathcal{T}_3^2$ w/$\text{TRS}_{0.6}$	**1.57**	3.13	**2.34***	**6.01***	4.35	**2201.03***	**3.3** *	**8.11***
				$\mathcal{E}_{\text{cx}}$				
GSGP	0.75	2.77	38.76	16.14	9.62	2407.62	13.63	10.58
$\mathcal{T}_1^2$ w/RKS	0.41	2.56	39.45	17.24	10.65	2421.11	13.67	10.93
$\mathcal{T}_2^2$ w/RKS	**0.29**	2.83	38.76	20.79	11.5	2401.69	13.86	10.97
$\mathcal{T}_3^2$ w/RKS	**0.26***	2.84	38.74	27.19	12.09	2421.87	14.09	11.09
$\mathcal{T}_1^2$ w/$\text{TRS}_{0.6}$	1.06	2.88	38.77	16.25	9.42	2384.49	13.4	10.51
$\mathcal{T}_2^2$ w/$\text{TRS}_{0.6}$	**0.37**	2.57	38.65	17.61	11.25	2404.46	13.89	10.93
$\mathcal{T}_3^2$ w/$\text{TRS}_{0.6}$	**0.28**	2.82	39.01	21.95	10.84	2415.91	13.95	10.89
				$\mathcal{E}_{\text{mut}}$				
GSGP	1.48	2.08	2.66	6.53	3.8	2278.42	4.11	8.68
$\mathcal{T}_1^2$ w/RKS	2.17	2.48	**2.58**	**6.37**	**3.69**	**2244.66**	**3.93**	**8.62**
$\mathcal{T}_2^2$ w/RKS	2.44	3.57	**2.48**	**6.1**	3.66	**2195.85**	**3.42**	**8.33**
$\mathcal{T}_3^2$ w/RKS	2.78	4.62	**2.4** *	**6.02***	3.81	**2172.03***	**3.26***	**8.27***
$\mathcal{T}_1^2$ w/$\text{TRS}_{0.6}$	1.63	1.93	2.69	6.54	3.79	**2273.45**	4.15	8.7
$\mathcal{T}_2^2$ w/$\text{TRS}_{0.6}$	2.54	2.83	**2.5**	**6.23**	3.72	**2214.63**	**3.59**	**8.39**
$\mathcal{T}_3^2$ w/$\text{TRS}_{0.6}$	2.75	3.58	**2.44***	**6.06***	3.7	**2186.87**	**3.41***	**8.33***

neighborhood, typically fails to produce results significantly better than those from GSGP.

Larger neighborhoods generally yield more consistent performance. While the best results come from $\mathcal{T}_3^2$, which has a relatively large neighborhood, a radius of 2 still consistently outperforms GSGP. Additionally, increasing the population size seems to enhance the likelihood that cGSGP outperforms the baseline, even when using a small neighborhood radius. This indicates that having a sufficiently large population is likely crucial for leveraging the cellular-based selection on the toroidal grid effectively.

Table 7.4: Table of the median best fitness for cGSGP on the test set and all considered topologies.

	400 as population size				250 generations			
Method	V14	KJ6	ARF	CNC	SLM	TXC	YCH	PRK
				$\mathcal{E}_{\text{cx+mut}}$				
GSGP	3.88	1.46	3.83	7.53	4.16	2375.11	5.85	9.07
$\mathcal{T}_1^2$ w/RKS	**2.16**	**0.94**	3.98	7.83	3.92	**2244.14**	6.44	9.26
$\mathcal{T}_2^2$ w/RKS	**0.93**	**0.89**	**3.28**	**7.12**	**3.83**	**2188.79**	**5.32**	**8.9**
$\mathcal{T}_3^2$ w/RKS	**0.89**	1.24	**2.9** *	**6.87***	**3.87**	**2201.99**	**4.45***	**8.71***
$\mathcal{T}_1^2$ w/$\text{TRS}_{0.6}$	9.43	1.3	4.53	8.05	4.11	2302.64	6.93	9.3
$\mathcal{T}_2^2$ w/$\text{TRS}_{0.6}$	**1.02**	**1.04**	**3.38**	**7.23**	**3.75**	**2274.06**	**5.45**	**8.99**
$\mathcal{T}_3^2$ w/$\text{TRS}_{0.6}$	**0.73***	1.04	**3.03***	**7.0** *	**3.77**	**2228.65***	**4.85***	**8.77***
				$\mathcal{E}_{\text{cx}}$				
GSGP	6.17	1.4	16.62	11.66	6.24	2372.28	11.42	10.04
$\mathcal{T}_1^2$ w/RKS	**1.84**	**0.85**	17.31	12.57	6.27	**2279.96**	11.71	10.3
$\mathcal{T}_2^2$ w/RKS	**0.31**	0.99	17.41	13.45	6.87	**2231.01**	12.0	10.35
$\mathcal{T}_3^2$ w/RKS	**0.28**	1.12	16.93	13.62	7.19	**2229.27**	12.1	10.33
$\mathcal{T}_1^2$ w/$\text{TRS}_{0.6}$	4.19	1.32	17.24	11.7	6.5	**2290.9**	11.54	10.04
$\mathcal{T}_2^2$ w/$\text{TRS}_{0.6}$	**0.37**	**0.97**	16.48	12.59	6.54	**2291.52**	11.84	10.3
$\mathcal{T}_3^2$ w/$\text{TRS}_{0.6}$	**0.28***	1.16	16.77	13.22	6.83	**2258.57**	11.99	10.35
				$\mathcal{E}_{\text{mut}}$				
GSGP	0.83	0.92	4.55	8.68	3.96	2232.81	6.65	9.44
$\mathcal{T}_1^2$ w/RKS	0.86	0.98	**4.46**	**8.6**	**3.82**	**2228.39**	6.86	9.47
$\mathcal{T}_2^2$ w/RKS	1.02	1.19	**3.47**	**7.6**	**3.8**	**2215.58**	**5.61**	**9.2**
$\mathcal{T}_3^2$ w/RKS	1.26	1.44	**3.12***	**7.1** *	**3.73**	**2207.89**	**5.09***	**9.01***
$\mathcal{T}_1^2$ w/$\text{TRS}_{0.6}$	0.82	0.92	5.02	8.96	3.88	2231.76	7.04	9.58
$\mathcal{T}_2^2$ w/$\text{TRS}_{0.6}$	0.88	1.18	**3.67**	**7.77**	**3.71**	**2221.74**	**5.94**	**9.23**
$\mathcal{T}_3^2$ w/$\text{TRS}_{0.6}$	1.08	1.35	**3.24***	**7.3** *	**3.68**	**2212.48**	**5.14***	**9.12***

Based on an analysis of the results for $\mathcal{E}_{\text{cx}}$ and $\mathcal{E}_{\text{mut}}$, we can conclude that the mutation operator on its own proves to be more advantageous than the crossover operator in the context of our approach. Specifically, when $\mathcal{E}_{\text{mut}}$ is utilized, our method tends to outperform the baseline more frequently, whereas the difference is less pronounced when $\mathcal{E}_{\text{cx}}$ is applied. This suggests that the mutation operator gains more from the proposed selection mechanism than the crossover operator does.

The cGSGP method using the $\mathcal{E}_{\text{cx}}$ combination outperforms the baseline only in the synthetic datasets (V14 and KJ6) and in the TXC dataset. Conversely, cGSGP with the $\mathcal{E}_{\text{mut}}$ combination surpasses the baseline in all datasets except the synthetic ones.

Table 7.5: Table of the median best fitness for cGSGP on the test set and all considered topologies.

	900 as population size				111 generations			
Method	V14	KJ6	ARF	CNC	SLM	TXC	YCH	PRK
				$\mathcal{E}_{\text{cx+mut}}$				
GSGP	97.29	1.46	4.77	8.52	4.42	2431.04	7.01	9.41
$\mathcal{T}_1^2$ w/RKS	**8.07**	1.01	5.5	9.18	4.3	**2298.56**	7.88	9.6
$\mathcal{T}_2^2$ w/RKS	**0.78**	**0.69**	**4.32**	**8.41**	**4.11**	**2223.62**	**6.75**	**9.35**
$\mathcal{T}_3^2$ w/RKS	**0.63***	**0.7**	**3.86***	**8.05***	**3.94***	**2184.18***	**6.11***	**9.19***
$\mathcal{T}_1^2$ w/$\text{TRS}_{0.6}$	204.42	1.94	6.07	9.28	4.47	**2304.14**	8.11	9.64
$\mathcal{T}_2^2$ w/$\text{TRS}_{0.6}$	**1.46**	**0.87**	**4.58**	8.49	4.24	**2206.71**	6.95	9.41
$\mathcal{T}_3^2$ w/$\text{TRS}_{0.6}$	**0.8** *	**0.84**	**4.03***	**8.11***	**3.91***	**2191.4** *	**6.41***	**9.28***
				$\mathcal{E}_{\text{cx}}$				
GSGP	25.91	1.49	8.06	10.56	5.44	2344.68	9.94	9.92
$\mathcal{T}_1^2$ w/RKS	**1.8**	**0.86**	10.26	11.14	5.9	**2258.13**	10.67	10.05
$\mathcal{T}_2^2$ w/RKS	**0.37**	**0.84**	10.08	12.04	6.04	**2198.97**	10.86	10.11
$\mathcal{T}_3^2$ w/RKS	**0.28***	**0.71**	9.77	12.4	5.96	**2205.84**	10.97	10.15
$\mathcal{T}_1^2$ w/$\text{TRS}_{0.6}$	132.09	1.43	9.25	10.57	5.61	**2317.25**	10.14	9.98
$\mathcal{T}_2^2$ w/$\text{TRS}_{0.6}$	**0.96**	**0.78**	9.28	11.19	5.86	**2253.37**	10.63	10.07
$\mathcal{T}_3^2$ w/$\text{TRS}_{0.6}$	**0.34***	**0.73**	9.36	11.93	5.85	**2200.64**	10.58	10.1
				$\mathcal{E}_{\text{mut}}$				
GSGP	0.58	0.83	8.56	11.35	4.38	2195.01	7.78	9.78
$\mathcal{T}_1^2$ w/RKS	0.64	0.84	8.83	**11.26**	4.3	**2193.37**	7.95	9.81
$\mathcal{T}_2^2$ w/RKS	0.8	0.76	**6.5**	**9.7**	**3.97**	**2190.31**	**7.15**	**9.62**
$\mathcal{T}_3^2$ w/RKS	0.89	0.75	**5.25***	**8.94***	**3.71***	**2188.36**	**6.5** *	**9.48***
$\mathcal{T}_1^2$ w/$\text{TRS}_{0.6}$	0.6	0.78	9.68	11.81	4.52	2195.13	8.1	9.89
$\mathcal{T}_2^2$ w/$\text{TRS}_{0.6}$	0.68	0.78	**7.01**	**10.13**	**3.89**	**2192.04**	**7.26**	**9.68**
$\mathcal{T}_3^2$ w/$\text{TRS}_{0.6}$	0.66	0.85	**5.72***	**9.29***	**3.69***	**2189.93**	**6.73***	**9.55***

Overall, the standard $\mathcal{E}_{\text{cx+mut}}$ combination continues to deliver the most consistent performance across the board.

7.4.2 Best Individuals Fitness Distribution

We show the box-plots for 100 as population size and 1000 generations, with $\mathcal{E}_{\text{cx+mut}}$ as exploration pipeline, regarding the RMSE on the test set of the best overall individuals in the last generation.

Fig. 7.2 displays the outcomes for the synthetic V14 dataset. The graph indicates

that the cGSGP approach outperforms the baseline for this dataset. Notably, the RKS strategy emerges as the most reliable, consistently delivering better results than the baseline.

Fig. 7.3 illustrates the results for the synthetic KJ6 dataset. In this scenario, cGSGP does not surpass the baseline, indicating that the proposed selection strategies do not offer an advantage for this particular problem when using a small population size. Additionally, the results suggest that employing a larger radius with the RKS strategy may even lead to worse performance for cGSGP.

Fig. 7.4 displays the results for the ARF dataset. As shown, all cGSGP variants outperform GSGP, except for the two-dimensional torus $\mathcal{T}_1^2$, which does not achieve statistically significant improvements regardless of the selection method used. It can also be noted that $\mathsf{TRS}_{0.6}$ results in slightly lower errors, though the difference is marginal. The largest neighborhood, $\mathcal{T}_3^2$, consistently produces the best results among all cGSGP methods, independently of the selection strategy applied.

The results for the CNC dataset, shown in Fig. 7.5, reveal how various topologies impact result quality. For $\mathcal{T}_1^2$, performance does not show statistically significant improvements over GSGP. However, as the neighborhood size increases, there is a clear enhancement in solution quality, regardless of the selection method used.

Similar to the CNC dataset, the results for the SLM dataset, as shown in Fig. 7.6, indicate that GSGP and cGSGP yield comparable performance. Specifically, the results for two different topologies do not show a statistically significant improvement over GSGP.

Fig. 7.7 displays the results for the TXC dataset. It shows that, although there is a minor improvement in solution quality as the neighborhood size increases, the $\mathcal{T}_3^2$ topology consistently delivers the best performance among all cGSGP configurations.

Fig. 7.8 shows the results for the YCH dataset, reinforcing the trend seen in other datasets. The use of local-only interactions leads to better performance compared to GSGP, and the quality of the solutions improves further with larger neighborhoods.

As with many other datasets, the results for the PRK dataset, shown in Fig. 7.9, indicate that a cGSGP-based approach with a sufficiently large radius is essential for outperforming GSGP, irrespective of the selection strategy used.

7.4.3 Convergence Rates

Fig. 7.10 shows the convergence rates of both GSGP and cGSGP with $\mathcal{E}_{\text{cx+mut}}$ and RKS, along with a population size of 100 and 1000 generations. For each generation, we plot the fitness evaluated on the test set of the best-so-far individual (the median across the repetitions), with shaded area denoting the inter-quartile range.

Line plots reveal a negative trend for the two synthetic datasets, particularly noticeable for KJ6, affecting both the baseline and cGSGP approaches. In these cases, test fitness tends to improve with more generations, especially evident for $\mathcal{T}_2^2$ and $\mathcal{T}_3^2$ in KJ6. Conversely, the proposed method consistently outperforms the baseline in the V14 dataset. However, it seems to reach a plateau after the final generation, indicating possible over-fitting for all methods across both synthetic datasets.

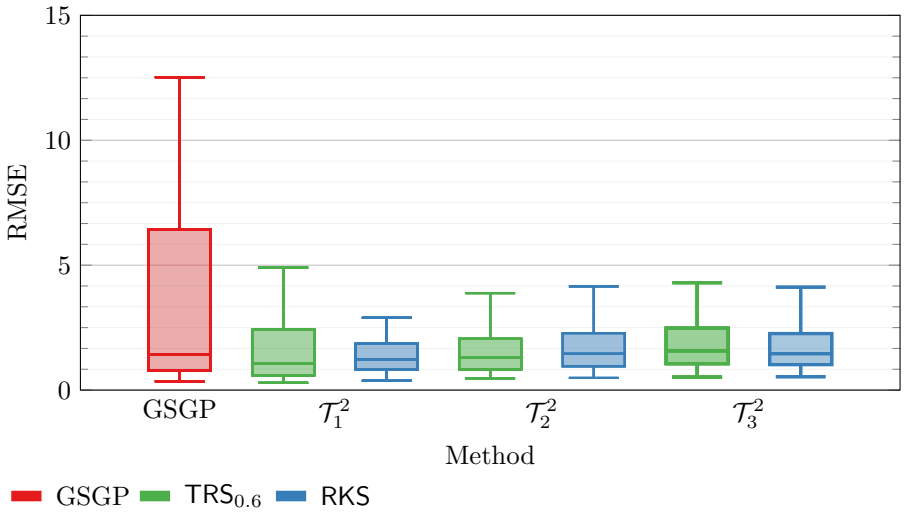

Figure 7.2: Test results on the V14 dataset. The box-plot shows the distribution of RMSE at the last generation.

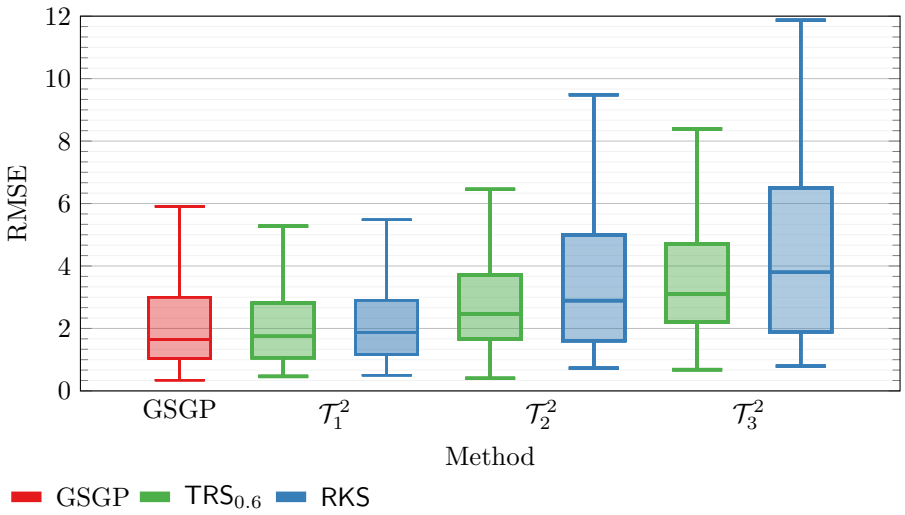

Figure 7.3: Test results on the KJ6 dataset. The box-plot shows the distribution of RMSE at the last generation.

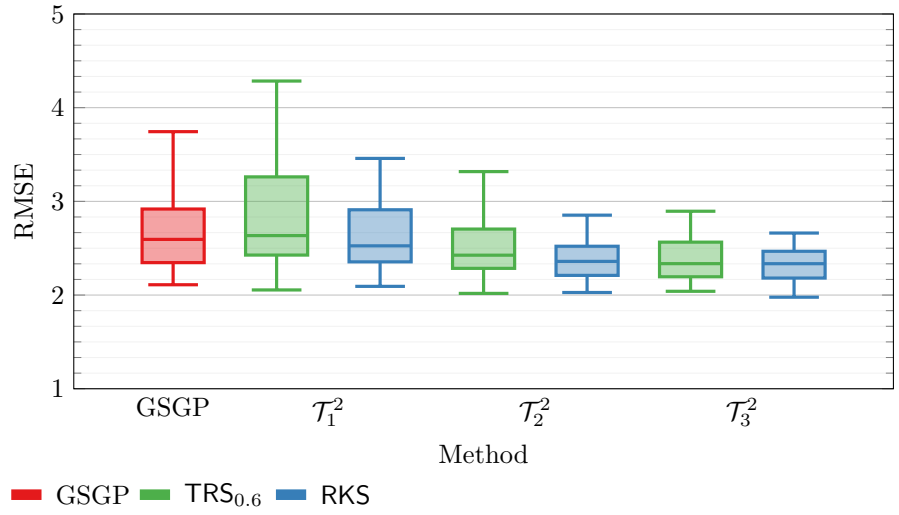

Figure 7.4: Test results on the ARF dataset. The box-plot shows the distribution of RMSE at the last generation.

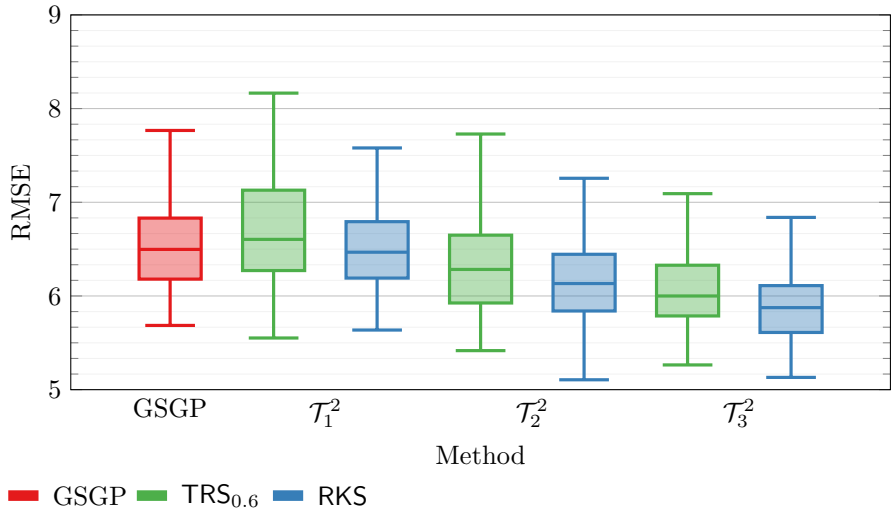

Figure 7.5: Test results on the CNC dataset. The box-plot shows the distribution of RMSE at the last generation.

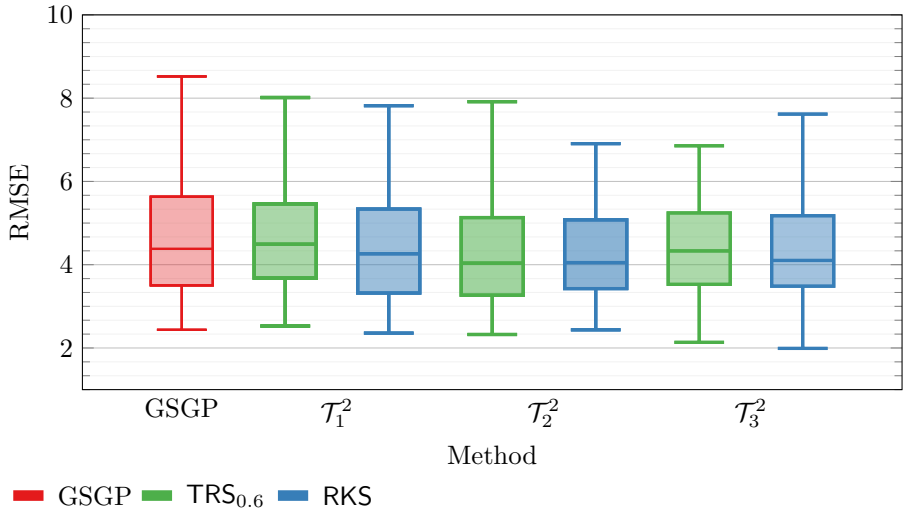

Figure 7.6: Test results on the SLM dataset. The box-plot shows the distribution of RMSE at the last generation.

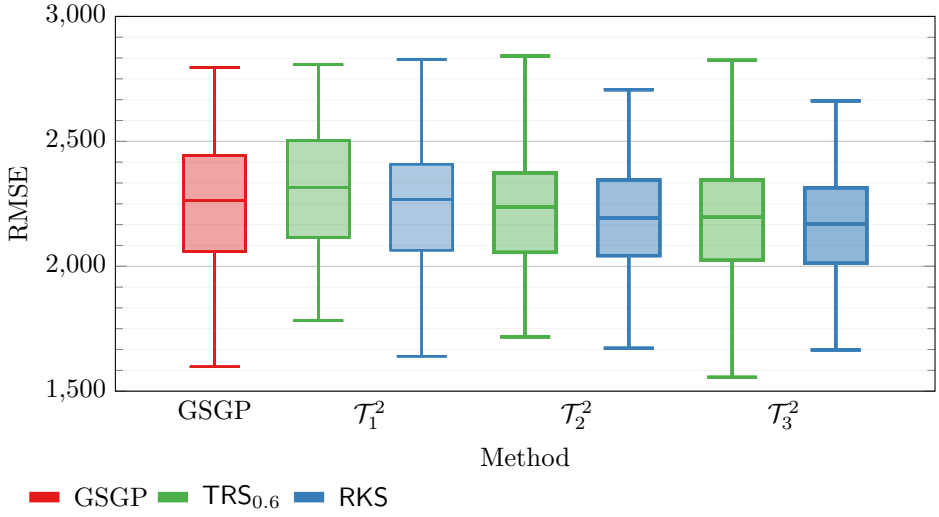

Figure 7.7: Test results on the TXC dataset. The box-plot shows the distribution of RMSE at the last generation.

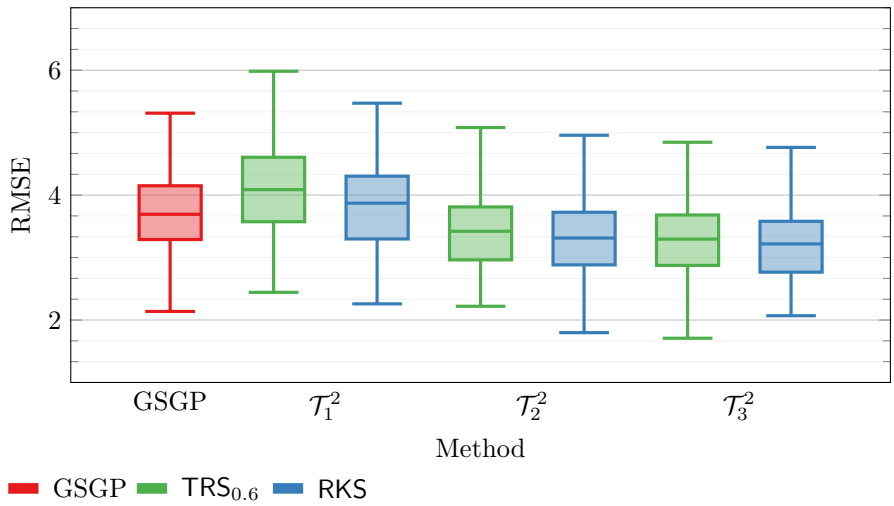

Figure 7.8: Test results on the YCH dataset. The box-plot shows the distribution of RMSE at the last generation.

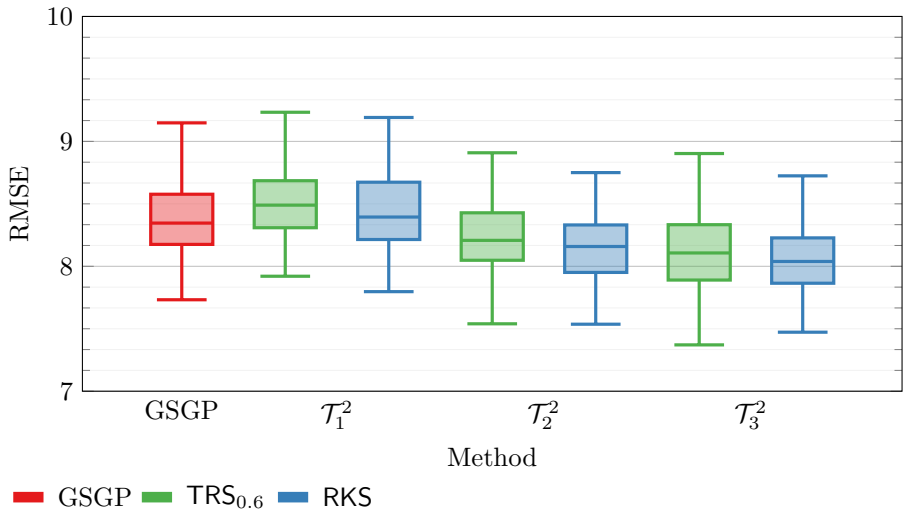

Figure 7.9: Test results on the PRK dataset. The box-plot shows the distribution of RMSE at the last generation.

For the remaining datasets, all methods show either a continuous decline in fitness (TXC, YCH, and PRK) or a sharp decrease in the initial generations followed by a plateau (ARF, CNC, and SLM). In these cases, GSGP and $\mathcal{T}_1^2$ generally exhibit similar patterns. However, cGSGP methods with a larger radius often display a more pronounced downward trend.

It appears that the evolutionary process benefits from a cellular-inspired selection mechanism, particularly when using a large radius. This method allows for the optimization process to achieve better results than traditional GSGP. Additionally, the improvement seems to be consistent regardless of the numbers of generations.

7.4.4 Population Fitness Distribution

After evaluating the performance of the proposed methods over the generations in terms of the best solutions found, we now focus on analyzing the fitness distribution of the entire population on the training set. To illustrate this, Fig. 7.11 presents

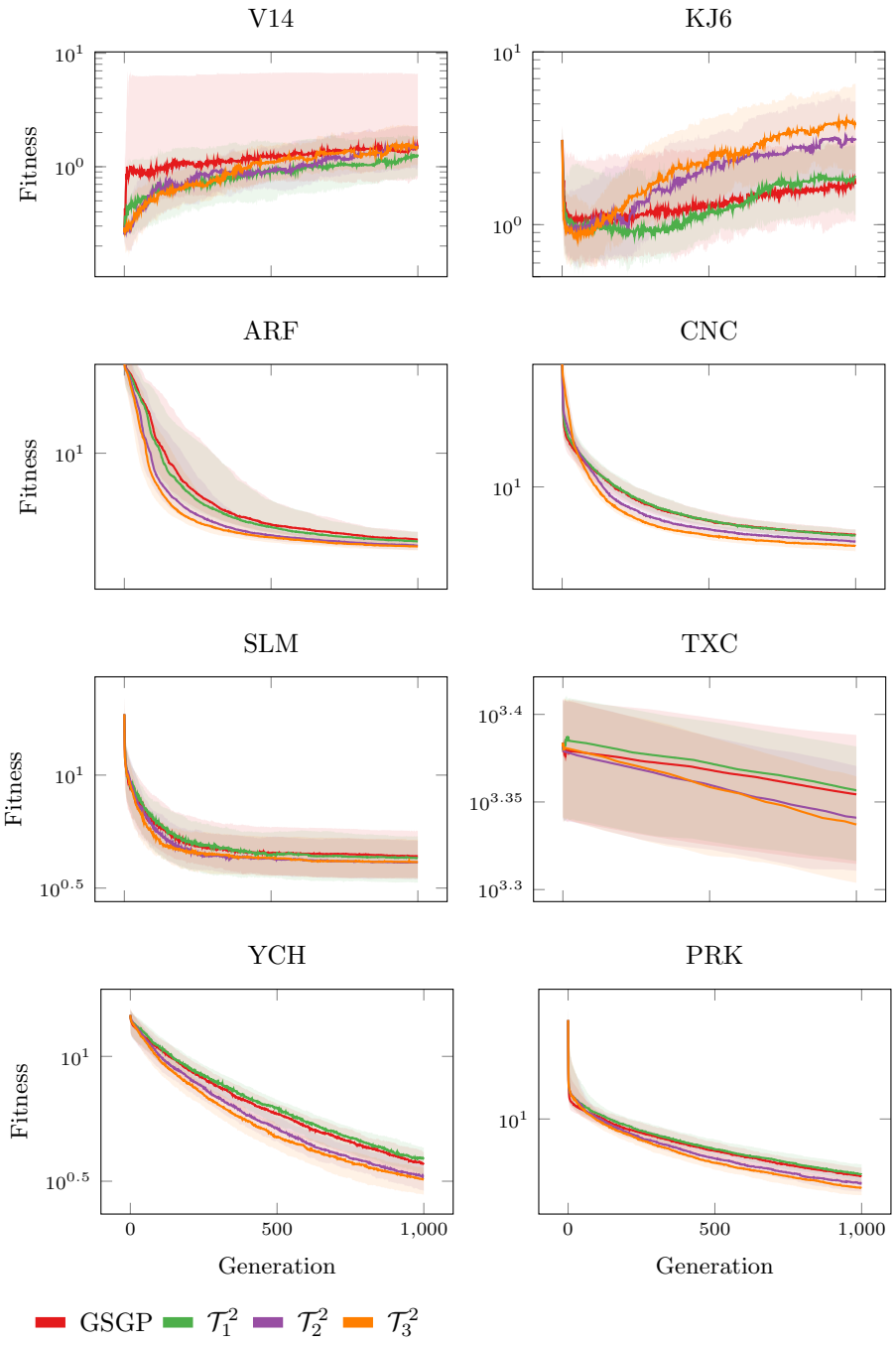

Figure 7.10: Trend of test RMSE of the best individual. The plot is based on $\mathcal{E}_{\text{cx+mut}}$ and RKS strategy, with 100 as population size and 1000 generations.

the median training fitness, where $\mathcal{E}_{\text{cx+mut}}$ and RKS are utilized with a population size of 100 across 1000 generations.

For each generation, the plot shows the median fitness value of all individuals in the population evaluated on the training set. The shaded area denotes the interquartile range, providing a measure of the fitness variability within the population for that generation.

For nearly all datasets tested, we observe a consistent reduction in the median RMSE. This pattern holds true even for the two synthetic datasets, which, as suggested by the convergence rate analysis, exhibit overfitting-related issues. Regarding

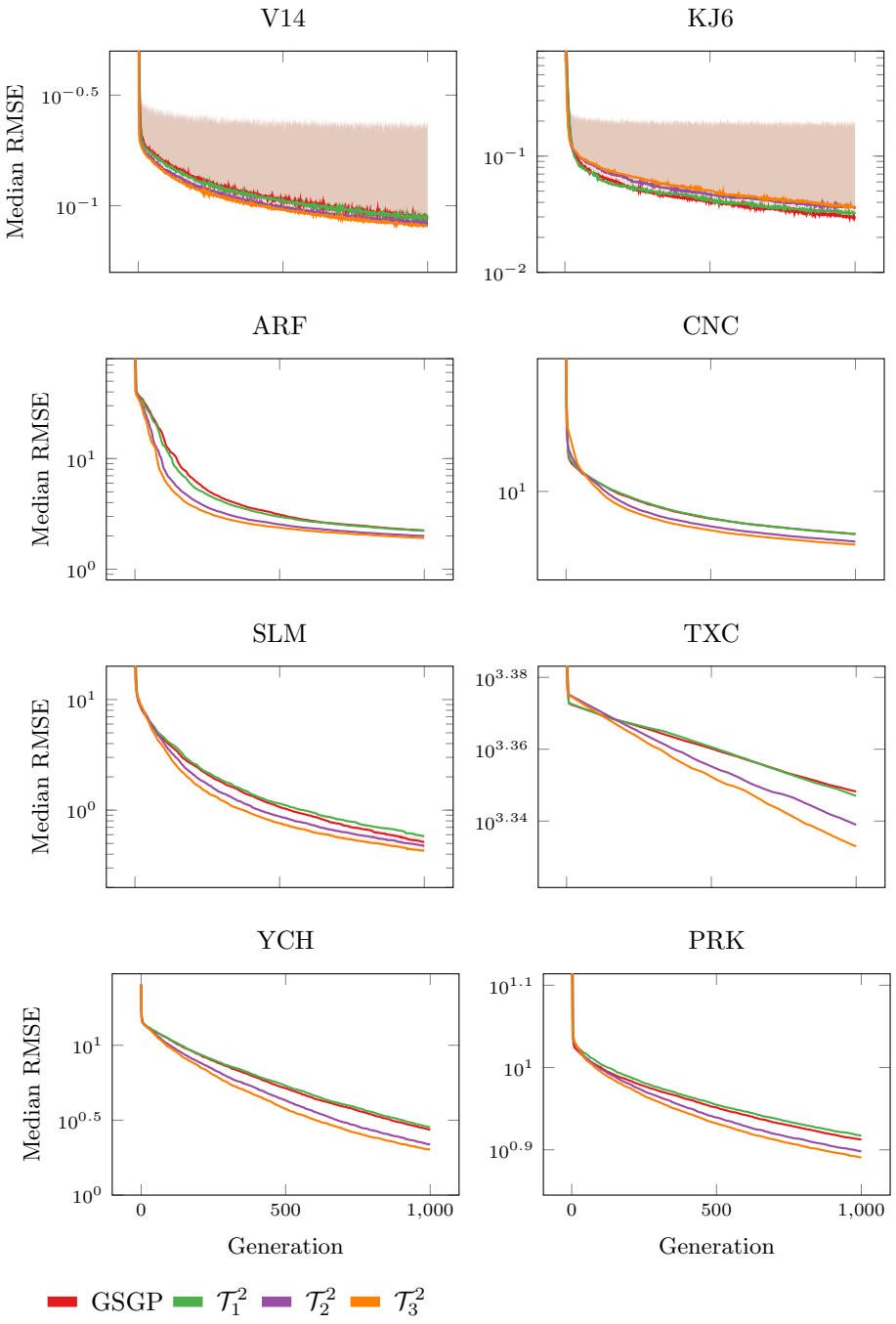

Figure 7.11: Trend of median train RMSE of the population. The plot is based on $\mathcal{E}_{\text{cx+mut}}$ and RKS strategy, with 100 as population size and 1000 generations.

the real-world datasets, the median RMSE on the training set aligns with the trends highlighted in the line-plots of the previous section, where TXC shows a downward trend with negative concavity for both $\mathcal{T}_2^2$ and $\mathcal{T}_3^2$. Furthermore, as previously noted, we observe a plateau after a few generations in the ARF and CNC datasets.

While analyzing the population's fitness distribution on the training set could provide insights into the diversity-preserving mechanisms of the proposed methods, the results show little to no variation in training fitness, except for the synthetic datasets and occasionally SLM. Therefore, this type of analysis offers limited information on population diversity, which will be explored in more depth in the following section.

7.4.5 Diversity

Assessing the diversity of a population over generations is particularly challenging in the context of GSGP, largely because analyzing the syntactical structure of trees can be problematic since syntactically different individuals will always produce different results from GSOs and the tree representation of a GSGP individual grows exponentially in size. Consequently, we address this challenge by employing a measure that evaluates the semantic distance between individuals to analyze population diversity.

As already described in Sec. 7.1, an individual evaluated on the training set produces a vector of predictions representing the semantic vector of the individual. Building on that, Global Moran's I (I) [342, 343] can be calculated on the semantic vectors of a given population and adopted as diversity measure. I is a measure of spatial auto-correlation [344], assessing similarity or dissimilarity of elements located in neighboring locations:

$$I = \frac{M}{W} \frac{\sum_{i=1}^{M} \sum_{j=1}^{M} \mathbf{w}_{ij}(\mathbf{v}^i - \bar{\mathbf{v}})(\mathbf{v}^j - \bar{\mathbf{v}})}{\sum_{i=1}^{M} (\mathbf{v}^i - \bar{\mathbf{v}})^2}$$

where M is the population size, $\mathbf{w}_{ij} \in \mathbb{R}$ is the value located at the i-th row and the j-th column of the matrix $\mathbf{w} \in \mathbb{R}^{M \times M}$, $\mathbf{w}_{ii} = 0$ for $1 \leq i \leq M$, $W = \sum_{i=1}^{M} \sum_{j=1}^{M} \mathbf{w}_{ij}$, $\mathbf{v}^i$ is the semantic vector of the i-th individual, and $\bar{\mathbf{v}}$ is the mean of the semantic vectors of all the individuals in the population. In our case, $\mathbf{w}_{ij} = 1$ if $i \neq j$ and the j-th individual belongs to the neighborhood of the i-th individual[3], otherwise $\mathbf{w}_{ij} = 0$.

I takes values from $[-1, 1]$, where values close to 1 indicate positive spatial auto-correlation, while values close to -1 indicate negative spatial auto-correlation. Values near 0 indicate a spatial pattern that is hardly distinguishable from a random one. A high I means that similar semantic vectors are spatially clustered in the population (high intra-similarity), preserving diversity among different clusters (high inter-dissimilarity).

In the following line-plots, we show the trend of I with $\mathcal{E}_{\text{cx+mut}}$ and RKS, and we test all three combinations of population size and number of generations to highlight the interplay between population size and diversity.

Fig. 7.12 illustrates the trend of I for a population size of 100 over 1000 generations. For the V14 and KJ6 datasets, cGSGP methods show a flattening of I towards 0 after the initial generations, whereas the baseline consistently remains close to 0. In contrast, cGSGP methods generally demonstrate higher diversity compared to the baseline, which maintains I values near 0 throughout most generations. Diversity tends to decrease as the neighborhood radius increases, with $\mathcal{T}_1^2$ consistently showing the highest I values. Across most cGSGP methods, diversity diminishes over generations, although for TXC, it remains relatively stable after an initial adjustment period.

Fig. 7.13 depicts the I trend for a population size of 400 over 250 generations. As observed previously, both V14 and KJ6 datasets show a rapid convergence towards 0 across all cGSGP methods, while standard GSGP consistently maintains 0 as I

[3] In standard GSGP the entire population is a unique single neighborhood.

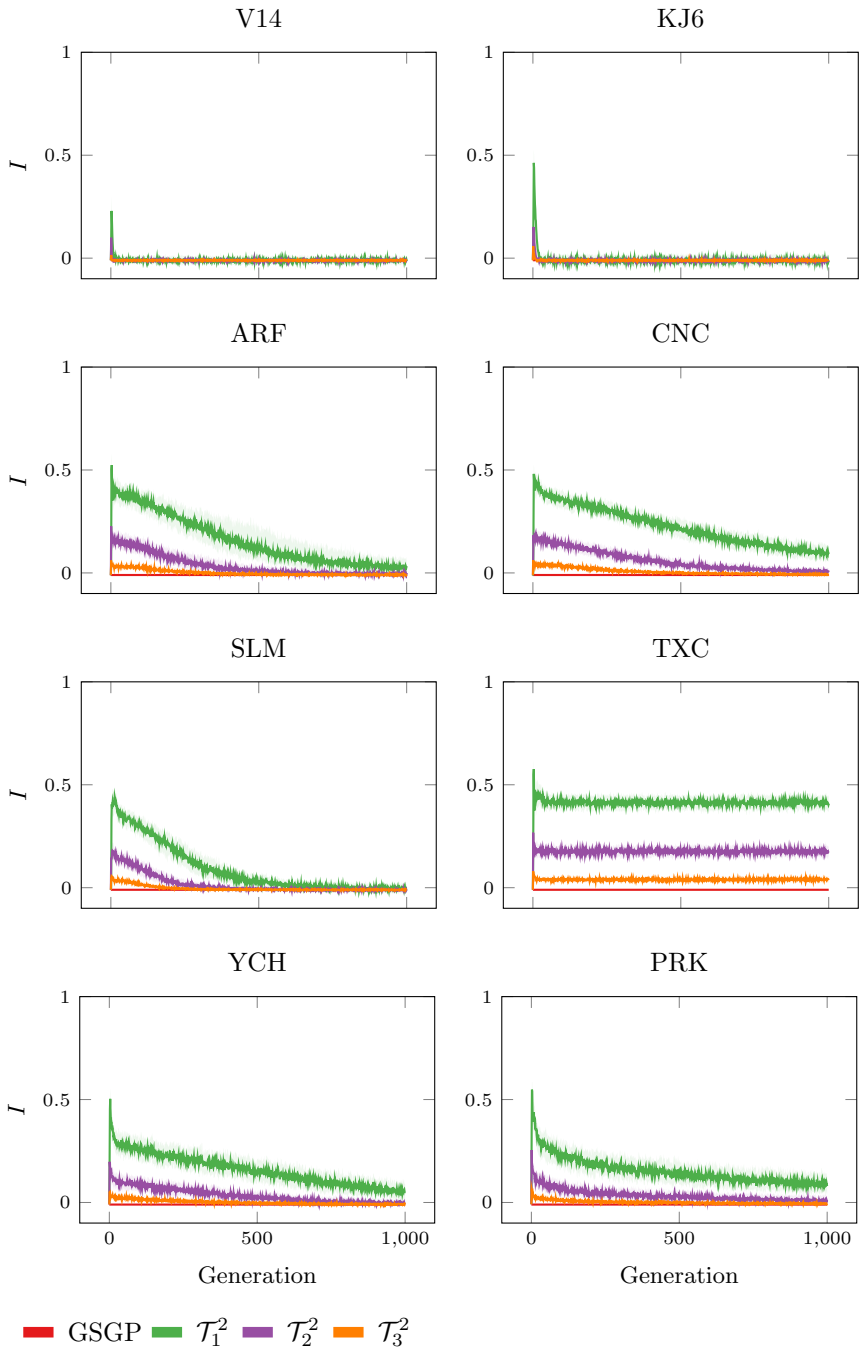

Figure 7.12: Trend of I. The plot is based on $\mathcal{E}_{\text{cx+mut}}$ and RKS strategy, with 100 as population size and 1000 generations.

throughout the evolution, similar to the other datasets. Among the various methods, $\mathcal{T}_1^2$ continues to exhibit the highest diversity. The trend in diversity across generations aligns with previous observations, with TXC showing nearly constant I values across all methods.

Fig. 7.14 illustrates the I trend for a population size of 900 over 111 generations. As with previous observations, both V14 and KJ6 datasets show a rapid decline towards 0 across all cGSGP methods. For all datasets, the baseline maintains a diversity level of zero. The overall pattern mirrors earlier findings: TXC exhibits nearly constant diversity for all methods, while other datasets show a reduction in

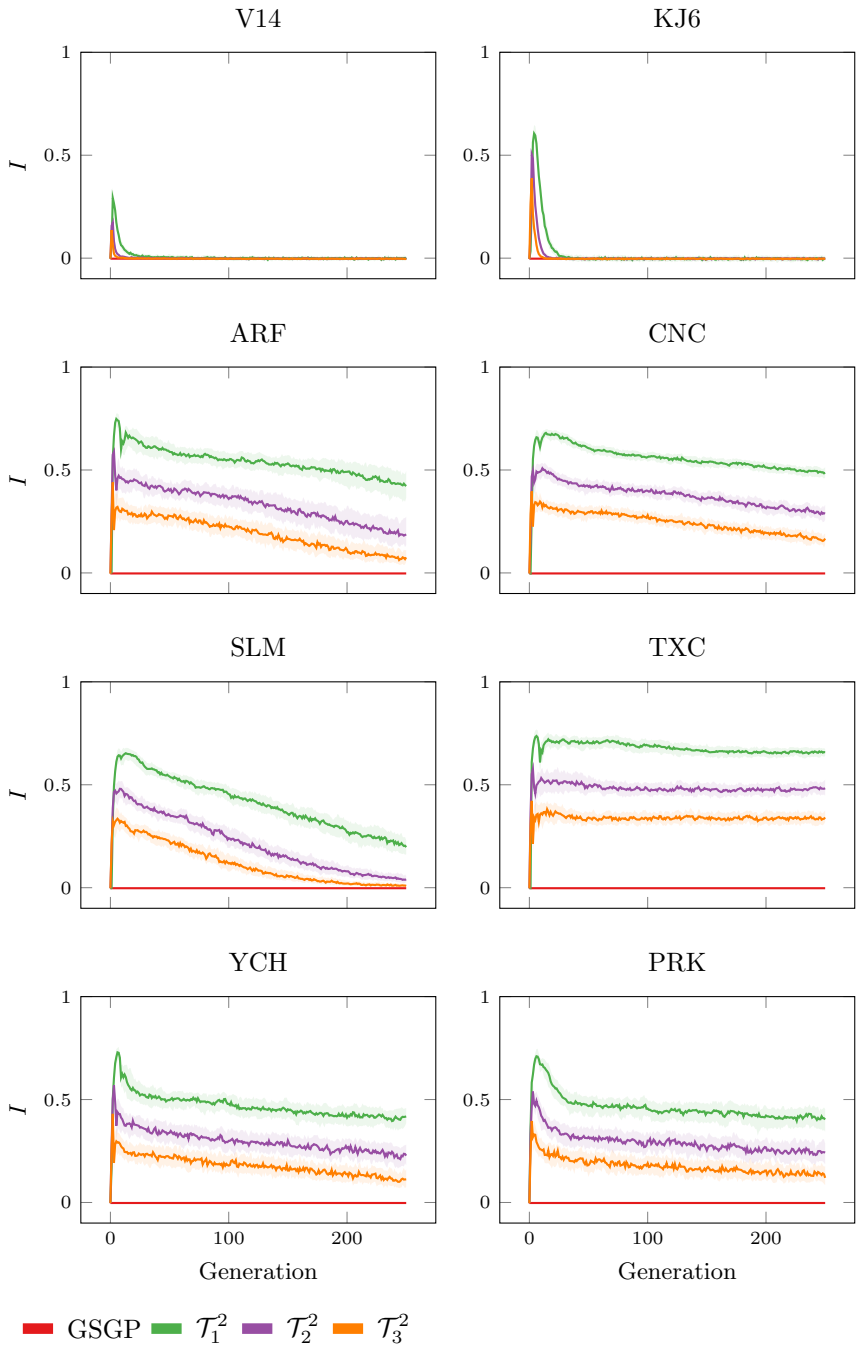

Figure 7.13: Trend of I. The plot is based on $\mathcal{E}_{\text{cx+mut}}$ and RKS strategy, with 400 as population size and 250 generations.

diversity over generations for all methods. Additionally, diversity increases with a decrease in the neighborhood radius.

It is observed that $\mathcal{T}_1^2$ consistently exhibits the highest diversity across all scenarios. This is due to local interactions being restricted to a small neighborhood, which slows the spread of advantageous individuals across generations compared to larger radii, leading to a higher I.

In summary, population size does not significantly impact the diversity trends across different methods, as similar patterns are observed regardless of the population size. In contrast, the neighborhood size plays a crucial role in maintaining spatial clusters

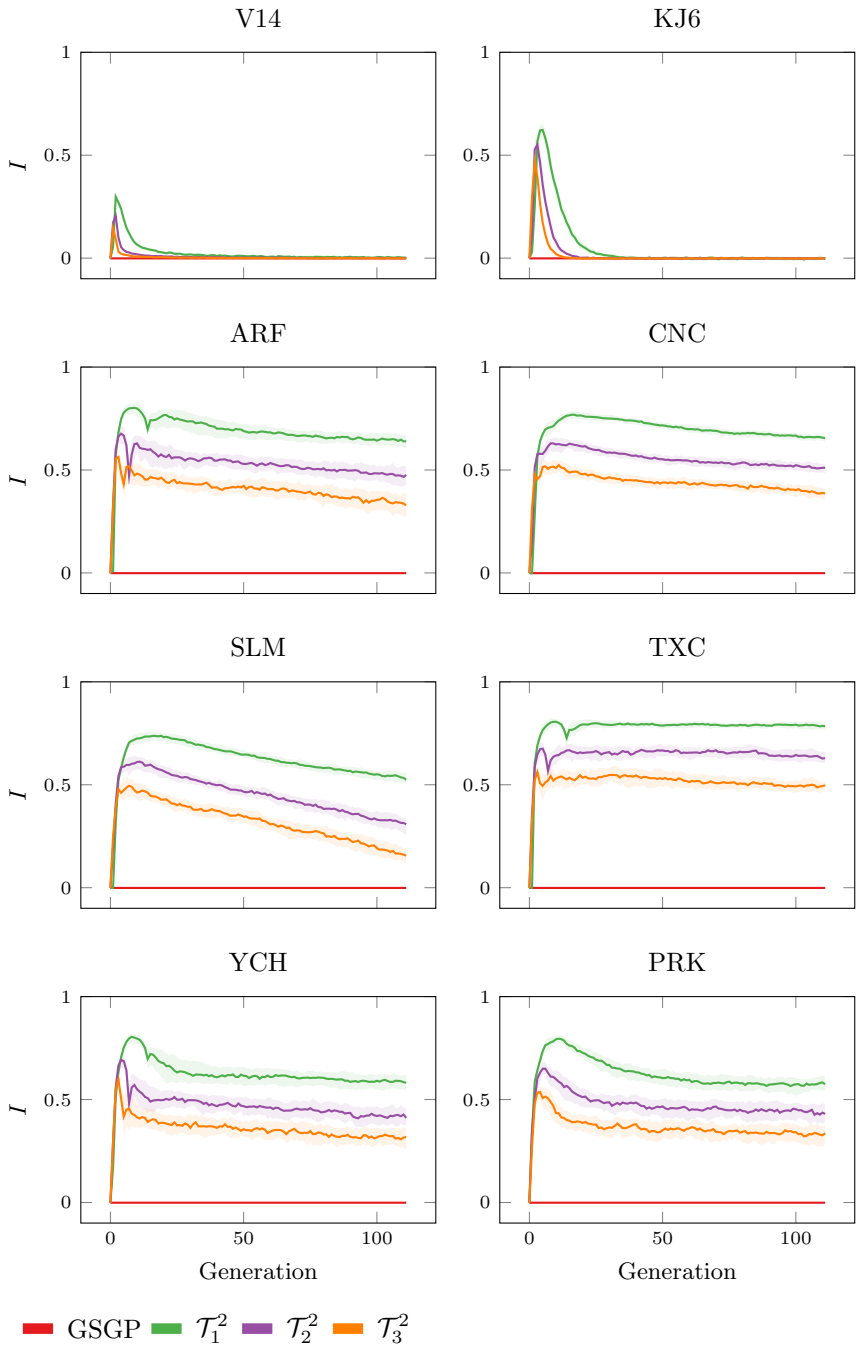

Figure 7.14: Trend of I. The plot is based on $\mathcal{E}_{\text{cx+mut}}$ and RKS strategy, with 900 as population size and 111 generations.

and preserving the semantic diversity of individuals.

7.4.6 Solution Size

In this last section, we investigate whether applying a CA-inspired communication topology to GSGP results in smaller solutions. This analysis is motivated by prior studies indicating that spatial structures can help control program growth and reduce solution size [265, 266].

A common property that measures the tree size is the number of nodes (ℓ). In

GSGP, this value increases exponentially with the number of generations. Hence, we focus on the base-10 logarithm in the number of nodes. Fig. 7.15 shows the distribution of the median $\log_{10}(\ell)$ in the last generation for all tested datasets and for all repetitions.

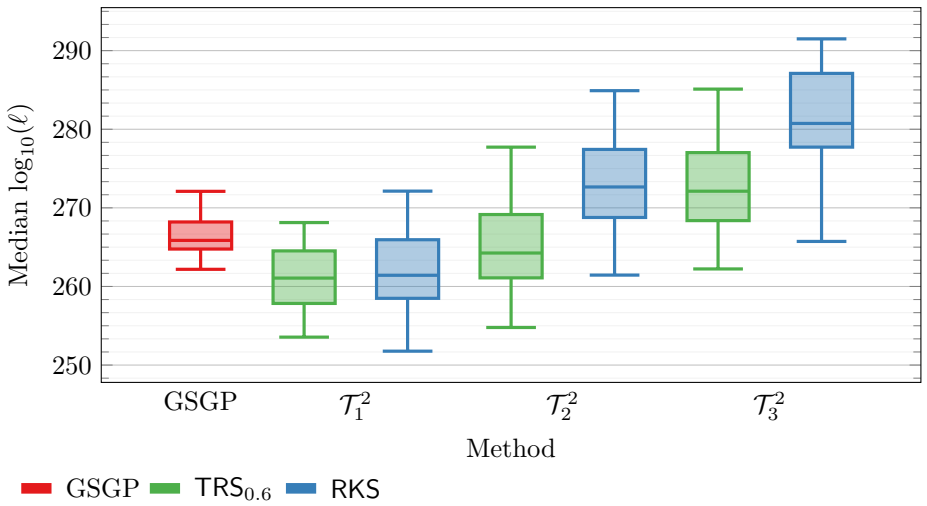

Figure 7.15: Distribution of the median $\log_{10}(\ell)$ computed on the population of the last generation for all the tested datasets and repetitions.

From the plot, it can be observed that $\mathcal{T}_1^2$ generally yields smaller solutions compared to GSGP, whereas $\mathcal{T}_2^2$ and $\mathcal{T}_3^2$ typically produce larger solutions, with the exception of $\mathcal{T}_2^2$ when using $\mathsf{TRS}_{0.6}$. This indicates that employing a cellular spatial structure with a small neighborhood effectively helps control program growth. However, unlike other methods, there is no consistent reduction in solution sizes across all neighborhood sizes when applying a cellular structure to GSGP.

Furthermore, when contrasting this result with the earlier analysis of convergence rates, the rise in the median size of individuals does not appear to negatively impact the quality of the generated solutions.

7.5 Discussion

The issue of premature convergence in GSGP likely stems from the absence of a built-in mechanism for preserving diversity. To mitigate this, we introduce a spatial structure to the population, enforcing local-only communication. This setup helps to extend the takeover time, promoting the maintenance of diversity. However, there is an inherent trade-off between the extent of local communication and the range of semantics that can be produced within a restricted neighborhood. Specifically, since semantic crossover results are constrained within the convex hull of the parent individuals, a smaller neighborhood limits the area in which new individuals can be generated, thereby restricting the range of possible semantics. As a result, generating new semantics may take more time, potentially slowing down the exploration process.

Through experimentation, we found that a smaller neighborhood radius enhances diversity among different neighborhoods but slows down the exploration process. Conversely, expanding the radius speeds up exploration but reduces overall diversity. Thus, striking a balance between maintaining diversity and ensuring efficient

exploration is crucial for optimal performance.

An analysis of Moran's I coefficient reveals that cGSGP promotes diversity across local neighborhoods during the initial phases of evolution, allowing for a more thorough exploration of the search space before dominant solutions spread throughout the population. This indicates that incorporating a spatial structure, like a cellular-based approach, can be advantageous in the context of GSGP as well.

In most instances, cGSGP-based methods outperform GSGP, particularly when using a radius of two or three in real-world datasets. This suggests that the neighborhood-based selection strategy leveraging local communication enhances GSGP performance. Additionally, an examination of convergence rates indicates that this advantage persists regardless of the total number of generations executed.

Chapter 8

Conclusion

This thesis has explored the application of Genetic Programming (GP) and its variations to address complex, high-impact problems across several domains. Through the in-depth study of non-linear Boolean function optimization, code generation enhancement, interpretable Machine Learning (ML) model discovery, and Symbolic Regression (SR), we have demonstrated the significant potential of Evolutionary Algorithms (EAs) in solving critical challenges. Alongside our contributions, we also propose several avenues for future research that can further refine and extend the methods presented in this thesis.

In the domain of non-linear Boolean function optimization for stream ciphers design, we introduced an approach leveraging Walsh transforms as solutions evolved by GP. This method proved capable of generating Boolean functions with high non-linearity and balancedness, both essential for secure cryptographic systems. Our results indicate that the biased uncertainty filling strategy was effective in guiding the evolutionary search toward balanced solutions, although achieving optimal non-linearity for n-variables Boolean functions with large values of n remains challenging, since the search space for highly secure Boolean functions is complex, making the discovery of optimal solutions difficult.

Future research should focus on comparing different exploration strategies within this search space, including our Walsh-based approach, to outline the most promising methods. Such studies should prioritize using a limited computational budget to identify the most effective techniques, which can later be employed with greater resources to discover the best possible solutions.

For Large Language Models (LLMs) code generation enhancement, we combined Genetic Improvement (GI) with the generation capabilities of LLMs to refine code generated by these models. Our experiments revealed that while LLMs can generate usable code, they often fail to correct their mistakes through re-prompting. GI, especially when guided by problem-specific specialized grammars and lexicase selection, consistently improved the correctness of LLM-generated code. This hybrid approach shows practical utility in improving code generation, particularly for simpler problems or when starting with smaller models.

Future research should focus on exploring the minimum number of test cases required during fitness evaluation to ensure reliable and accurate results. Quantifying the effort required by users when interacting with the system is essential for optimizing productivity. Additionally, new techniques for dynamically specializing grammars in GI could help expedite the exploration of the search space. One promising direction involves using a large grammar for initial parsing to increase the chances of producing valid solutions, followed by grammar reduction based on LLM outputs, which could shrink the search space for GI.

In the context of interpretable ML models, we introduced the ML-PIE framework,

a human-in-the-loop system combining multi-objective GP to evolve models that balance predictive accuracy (or other qualitative metrics) with user-defined interpretability. By incorporating user feedback into the GP process, our framework allowed for the dynamic tailoring of models to meet the subjective interpretability needs of different users. Although our system was tested mainly on SR problems, it is generalizable to other machine learnable problems, as long as the models can be numerically encoded for the interpretability estimation performed by a neural network and two models can be visualized and compared such that the user can easily select the most interpretable one.

The trade-offs between encoding complexity, user feedback, and computational resources represent critical aspects of the framework. More complex encodings allow the system to approximate intricate interpretability definitions but require substantial feedback and computational time, while simpler encodings are faster but less expressive.

A key challenge in high-stakes applications is the trade-off between accuracy and interpretability, and our framework effectively managed this balance. We observed that interpretability is highly dependent on the user's background and involvement, with ample feedback required to produce models aligned with individual user expectations. Our survey with real users indicated that a substantial amount of feedback is necessary for models generated by ML-PIE to be preferred.

Future research should apply the framework to domain-specific problems, involving experts who can provide more meaningful feedback in evolving interpretable models. Additionally, further studies should explore warm-up strategies and improved general representations to reduce user effort by decreasing the amount of feedback required by the interpretability estimator, as well as investigate whether domain-specific representations offer the most practical solutions in real-world applications.

For SR, we introduced a novel approach, named Cellular Geometric Semantic Genetic Programming (cGSGP), to mitigate the issue of premature convergence in standard Geometric Semantic Genetic Programming (GSGP) by incorporating a spatial structure into the population. This structure helped maintain greater diversity in local neighborhoods, facilitating more thorough exploration before convergence in the early phase of the evolution. Our experiments confirmed that the radius of local communication has a significant impact on both diversity and exploration speed, with a moderate radius providing the best balance between the two. In most cases, cGSGP outperformed traditional GSGP, demonstrating that neighborhood-based evolution is a promising enhancement to conventional GSGP techniques.

Future research in this area should focus on the development of scalable and efficient implementations of cGSGP, as the method local-only communication structure lends itself to distributed computation across multiple nodes. Further studies should also investigate the influence of topology on performance, the best methods for local selection, and the propagation of information within the population.

In conclusion, this thesis advances the field of EAs by addressing critical challenges in discrete optimization, cryptographic security, software correctness, model interpretability, and SR. Our contributions demonstrate the versatility and effectiveness of GP in solving a variety of complex problems. However, many open questions and potential improvements remain. Future research should focus on refining EAs

to enhance diversity preservation, comparing exploration strategies in non-linear Boolean function optimization, investigating dynamic grammar specialization in GI, and exploring scalable implementations of cGSGP. Additionally, future studies should prioritize quantifying the user effort required in human-in-the-loop systems, particularly in the context of Interpretable Machine Learning (IML), since high-stakes real-world applications urgently need models deployed in their production environments to be interpretable and understandable to more securely drive decision-making.

These efforts will undoubtedly drive further progress in this field, paving the way for more efficient and practical applications of GP in solving complex real-world problems across various scientific and technological domains.

Appendix A

Vernam Stream Cipher

Contents

Symmetric cryptography includes all those techniques applied to contexts where parties communicate over an unsafe channel by means of a shared private key that is employed for both encryption and decryption. A stream cipher is a symmetric cipher that encrypts and decrypts a message as a single block one bit at a time [71]. For the sake of simplicity, from now on we assume that a message is processed one bit at a time and is represented as a binary string of fixed length. Moreover, we are going to adopt the notation of mathematical functions to describe input and output of the mentioned algorithms because of its high expressiveness. However, we would like to stress that these algorithms are not functions in the mathematical term since, in some cases, the same input may lead to different outputs.

Let $\mathbb{B} = \{0,\ 1\}$ be the set containing the binary values and let $n > 0$ be the length of a given plain-text message $\mathbf{m} \in \mathbf{M}$, where $\mathbf{M} = \mathbb{B}^n$ is the set containing all possible plain-text messages of length n. Let $\mathbf{K} = \mathbb{B}^n$ be the set of all possible shared private keys and $\mathbf{C} = \mathbb{B}^n$ the set of all possible cipher-text messages. A symmetric cipher is a pair consisting of the following public and efficient (polynomial-time) algorithms: an encryption algorithm $E : \mathbf{M} \times \mathbf{K} \to \mathbf{C}$ and a decryption algorithm $D : \mathbf{C} \times \mathbf{K} \to \mathbf{M}$, so that $D(E(\mathbf{m}, \mathbf{k}), \mathbf{k}) = \mathbf{m}$ for each shared private key $\mathbf{k} \in \mathbf{K}$ and for each plain-text message $\mathbf{m} \in \mathbf{M}$.

A.1 One-Time Pad

The Vernam cipher [284], also known as One Time Pad (OTP), is a stream cipher where both encryption and decryption algorithms are implemented by using a XOR operation (denoted as $\oplus$).

Given two bits a and b, the XOR operation is defined as:

$$a \oplus b = \begin{cases} 0 & \text{if } a = b \\ 1 & \text{if } a \neq b \end{cases}$$

Moreover, it exhibits the following properties:

- Commutativity: $a \oplus b = b \oplus a$
- Associativity: $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

- Identity: $a \oplus 0 = a$
- Self-inverse: $a \oplus a = 0$

Given $\mathbf{a} \in \mathbb{B}^n$ and $\mathbf{b} \in \mathbb{B}^n$ and let $|$ be the concatenation operator, the XOR between $\mathbf{a}$ and $\mathbf{b}$ is defined as:

$$\mathbf{a} \oplus \mathbf{b} = \mathbf{a}_1 \oplus \mathbf{b}_1 \mid \mathbf{a}_2 \oplus \mathbf{b}_2 \mid \ \ldots \ \mid \mathbf{a}_n \oplus \mathbf{b}_n$$

where $\mathbf{a}_i$ ($\mathbf{b}_i$) with $i \in [1..n]$ is the i-th bit in $\mathbf{a}$ ($\mathbf{b}$).

In the OTP, $E(\mathbf{m}, \mathbf{k}) = \mathbf{m} \oplus \mathbf{k}$ and $D(\mathbf{c}, \mathbf{k}) = \mathbf{c} \oplus \mathbf{k}$. By using the properties of the XOR, it can be proved the following:

$$D(E(\mathbf{m}, \mathbf{k}), \mathbf{k}) = D(\mathbf{m} \oplus \mathbf{k}, \mathbf{k}) = (\mathbf{m} \oplus \mathbf{k}) \oplus \mathbf{k} = \mathbf{m} \oplus (\mathbf{k} \oplus \mathbf{k}) = \mathbf{m} \oplus \mathbf{0} = \mathbf{m}$$

where $\mathbf{0} \in \mathbb{B}^n$ is the n-sized binary string containing all zeros.

Furthermore, it can be proved that $\mathbf{k} = \mathbf{m} \oplus \mathbf{c}$. As a matter of fact, given a plain-text $\mathbf{m}$ and a cipher-text $\mathbf{c} = \mathbf{m} \oplus \mathbf{k}$, it holds that:

$$\mathbf{m} \oplus \mathbf{c} = \mathbf{m} \oplus (\mathbf{m} \oplus \mathbf{k}) = (\mathbf{m} \oplus \mathbf{m}) \oplus \mathbf{k} = \mathbf{0} \oplus \mathbf{k} = \mathbf{k}$$

Since OTP is a one-time key cipher, the key, which must be sampled randomly according to a uniform distribution, must be changed at every message exchange. Consider that a sender wants to send two plain-text messages $\mathbf{m}_1$ and $\mathbf{m}_2$ with the same key $\mathbf{k}$. By using the OTP, the sender produces two cipher-texts $\mathbf{c}_1 = \mathbf{m}_1 \oplus \mathbf{k}$ and $\mathbf{c}_2 = \mathbf{m}_2 \oplus \mathbf{k}$. A potential eavesdropper could then take $\mathbf{c}_1$ and $\mathbf{c}_2$ from the communication channel and perform $\mathbf{c}_1 \oplus \mathbf{c}_2$, which, as we can see, it is equal to $\mathbf{m}_1 \oplus \mathbf{m}_2$ if the secret key remains unchanged:

$$\mathbf{c}_1 \oplus \mathbf{c}_2 = (\mathbf{m}_1 \oplus \mathbf{k}) \oplus (\mathbf{m}_2 \oplus \mathbf{k}) = (\mathbf{m}_1 \oplus \mathbf{m}_2) \oplus (\mathbf{k} \oplus \mathbf{k}) = (\mathbf{m}_1 \oplus \mathbf{m}_2) \oplus \mathbf{0} = \mathbf{m}_1 \oplus \mathbf{m}_2$$

Knowing the result of $\mathbf{m}_1 \oplus \mathbf{m}_2$, the adversary can deduce, by using statistical-based analysis, secret parts of the original messages.

In [288], Shannon mathematically proved that the Vernam cipher satisfies the perfect secrecy property (i.e., the cipher-text must reveal no information, neither a single bit, about the plain-text), making this cipher confidentially safe against eavesdropping attacks [345]. Formally, a cipher (E, D) defined over $(\mathbf{K}, \mathbf{M}, \mathbf{C})$ has perfect secrecy if, for each $\mathbf{m}_1 \in \mathbf{M}$, $\mathbf{m}_2 \in \mathbf{M}$, and $\mathbf{c} \in \mathbf{C}$, it holds that

$$\Pr[E(\mathbf{m}_1, \mathbf{k}) = \mathbf{c}] = \Pr[E(\mathbf{m}_2, \mathbf{k}) = \mathbf{c}]$$

where the key $\mathbf{k}$ is sampled randomly according to a uniform distribution from $\mathbf{K}$.

It is easy to prove that OTP satisfies the perfect secrecy since $\Pr[E(\mathbf{m}, \mathbf{k}) = \mathbf{c}] = \frac{1}{|\mathbf{K}|}$ for each $\mathbf{m} \in \mathbf{M}$ and $\mathbf{c} \in \mathbf{C}$, with $\mathbf{k}$ sampled randomly according to a uniform distribution from $\mathbf{K}$. This statement holds since, as previously demonstrated with

$\mathbf{k} = \mathbf{m} \oplus \mathbf{c}$, given a plain-text message and a cipher-text message, there is only one key that enables us to compute the given cipher-text message from the given plain-text message.

To ensure secure communication, the key must be changed at every new message. This means that the involved parties must agree to a new private shared key every time a new message is crafted and needs to be forwarded. Moreover, this key needs to be at least as long as the message (necessary condition in order to let the perfect secrecy be, eventually, satisfied). This poses several complications in using this cipher in real-world applications where messages are typically long [345].

A.2 Pseudo-Random Generators

In practical cases, the Vernam cipher is combined with a Pseudo-Random Generator (PRG), i.e., a public, efficient (polynomial-time), and deterministic algorithm $G : \mathbb{B}^s \to \mathbb{B}^n$ where $s << n$ and $s > 0$ [285, 286]. A PRG gets a relatively short binary string as input that is typically randomly generated by using a uniform distribution and outputs a way longer binary string that seems randomly generated even if the underlying algorithm is deterministic. A PRG-based Vernam cipher is defined by $E_G(\mathbf{m}, \mathbf{k}) = \mathbf{m} \oplus G(\mathbf{k})$ and $D_G(\mathbf{c}, \mathbf{k}) = \mathbf{c} \oplus G(\mathbf{k})$. Let $\mathbf{K}' = \mathbb{B}^s$ and given a short key $\mathbf{k}' \in \mathbf{K}'$, E_G and D_G can be applied by using $\mathbf{k}'$ as key, provided that $\mathbf{k}'$ is sampled randomly according to a uniform distribution from $\mathbf{K}'$ and the key is changed at every new message.

The main advantage of this construction is allowing the parties involved in the communication to adopt short keys that can be easily exchanged. The disadvantage is its reliance on the safety properties of the PRG, which must be unpredictable to guarantee the safety of the cipher. Moreover, this cipher cannot satisfy the perfect secrecy property. Hence, PRG-based Vernam ciphers usually rely on less strict, but still valid, security definitions, such as the semantic security.

A.3 Semantic Security and Unpredictability

A symmetric cipher is semantically secure as regards confidentiality against eavesdropping attacks if it is infeasible for an adversary to distinguish between the encryptions of two chosen plain-texts, given the cipher-text and without knowledge of the secret key [346]. More formally, let (E, D) be a one-time key symmetric cipher. The cipher is semantically secure if, for any probabilistic polynomial-time adversary A, the advantage in the following game is negligible:

1. **Initialization**: A outputs two plain-text messages $\mathbf{m}_0$ and $\mathbf{m}_1$ of equal length and sends them to the cipher owner;
2. **Challenge**: a key $\mathbf{k}$ is generated at random by using a uniform distribution by the cipher owner and a random secret bit $b \in \{0, 1\}$ is chosen. The cipher-text $\mathbf{c} = E(\mathbf{m}_b, \mathbf{k})$ is computed and given to A;
3. **Guess**: A outputs a guess $b' \in \{0, 1\}$, indicating the plain-text that, according to A, generated $\mathbf{c}$.

The advantage of the adversary A is defined as:

$$\mathrm{Adv}_A^{\mathrm{SS}} = |\Pr[A(E(\mathbf{m}_0, \mathbf{k})) = 1] - \Pr[A(E(\mathbf{m}_1, \mathbf{k})) = 1]|$$

The one-time key symmetric cipher is semantically secure if $\mathrm{Adv}_A^{\mathrm{SS}}$ is negligible for any probabilistic polynomial-time adversary A.

A PRG-based Vernam cipher is semantically secure if the PRG is secure.

A PRG G is said to be unpredictable if, given the first n_τ bits of its output, it is computationally infeasible to predict the next bit with high probability.

Formally, given G, for $\mathbf{x} \in \mathbb{B}^s$ sampled at random with a uniform distribution, let $G(\mathbf{x}) = \mathbf{y}_1 \mid \mathbf{y}_2 \mid \ \ldots \ \mid \mathbf{y}_n$, where $\mathbf{y}_i$ with $i \in [1..n]$ is an individual output bit. G is predictable if, there exists a probabilistic polynomial-time adversary A and a position i with $i \in [1..n-1]$ such that:

$$\Pr\left[A(\mathbf{y}_1 \mid \mathbf{y}_2 \mid \ \ldots \ \mid \mathbf{y}_i) = \mathbf{y}_{i+1}\right] > \frac{1}{2} + \epsilon$$

where $\epsilon > 0$ is non-negligible. G is unpredictable if it is not predictable, meaning that, for each i, there exists no probabilistic polynomial-time adversary that can predict the $(i+1)$-th bit for non-negligible ϵ according to the aforementioned definition. The value of ϵ (as in the following definitions) depends upon the specific application domain and requirements.

The definition of unpredictability is strictly related to the definition of security in case of PRGs. The idea is that a PRG is secure if its output is indistinguishable from something that is truly random. Practically, the distribution of the outputs of the PRG calculated starting from binary strings sampled at random according to a uniform distribution from $\mathbb{B}^s$ is indistinguishable from the distribution of binary strings sampled at random according to a uniform distribution from $\mathbb{B}^n$ (recall that $s << n$).

In this scenario, a probabilistic polynomial-time adversary A is given a binary string. A outputs a single bit that tells whether the input binary string is random or not. Given a PRG G that outputs n-sized binary strings and a probabilistic polynomial-time adversary A that evaluates n-sized binary strings, the advantage of A is defined as:

$$\mathrm{Adv}_A^{\mathrm{PRG}} = |\Pr[A(G(\mathbf{k})) = 1] - \Pr[A(\mathbf{r}) = 1]|$$

where $\mathbf{k}$ is sampled at random according to a uniform distribution from $\mathbb{B}^s$ and $\mathbf{r}$ is sampled at random according to a uniform distribution from $\mathbb{B}^n$. The PRG G is secure if $\mathrm{Adv}_A^{\mathrm{PRG}}$ is negligible for any probabilistic polynomial-time adversary A.

Given a secure PRG G, it is possible to build a PRG-based Vernam stream cipher from G that is semantically secure. As a matter of fact, if G is secure, then the stream cipher derived from G is semantically secure, since for each probabilistic polynomial-time adversary A of the cipher, there exists a probabilistic polynomial-time adversary B of the PRG such that $\mathrm{Adv}_A^{\mathrm{SS}} \leq 2\mathrm{Adv}_B^{\mathrm{PRG}}$.

We show that if a PRG is predictable, then it is not secure. Suppose we have a probabilistic polynomial-time adversary A and a PRG G. With $\mathbf{x} \in \mathbb{B}^s$ sampled at

random with a uniform distribution, let $G(\mathbf{x}) = \mathbf{y}_1 \mid \mathbf{y}_2 \mid \ \ldots \ \mid \mathbf{y}_n$, where $\mathbf{y}_i$ with $i \in [1..n]$ is an individual output bit. Suppose it holds the following for a position i with $i \in [1..n-1]$:

$$\Pr\left[A(\mathbf{y}_1 \mid \mathbf{y}_2 \mid \ \ldots \ \mid \mathbf{y}_i) = \mathbf{y}_{i+1}\right] > \frac{1}{2} + \epsilon$$

where $\epsilon > 0$ is non-negligible. This A is an efficient algorithm that is capable to predict with high probability the $(i+1)$-th bit of the PRG output, given the first i bits of it (for some i). This PRG is predictable. We want to prove that this PRG is also not secure, meaning that there exists a probabilistic polynomial-time adversary B with a non-negligible advantage over G.

We define an adversary B as:

$$B(\mathbf{v}) = \begin{cases} 1 & \text{if } A(\mathbf{v}_1 \mid \mathbf{v}_2 \mid \ \ldots \ \mid \mathbf{v}_i) = \mathbf{v}_{i+1} \\ 0 & \text{else} \end{cases}$$

Given that $\mathbf{k}$ is sampled at random according to a uniform distribution from $\mathbb{B}^s$ and $\mathbf{r}$ is sampled at random according to a uniform distribution from $\mathbb{B}^n$, we can show that:

$$\mathrm{Adv}_B^{\mathrm{PRG}} = |\Pr[B(G(\mathbf{k})) = 1] - \Pr[B(\mathbf{r}) = 1]| > \epsilon$$

since $\Pr[B(G(\mathbf{k})) = 1] > \frac{1}{2} + \epsilon$ by definition (G is predictable) and $\Pr[B(\mathbf{r}) = 1] = \frac{1}{2}$ ($\mathbf{r}$ is truly random). In our hypothesis, ϵ is non-negligible, and thus B has a non-negligible advantage over G, meaning that G is not secure. Therefore, if a PRG is predictable, then it is not secure. As such, we can also state that if a PRG is secure, then it is unpredictable. Finally, it is also possible to prove that if a PRG is unpredictable then it is secure and, hence, we can conclude that a PRG is secure if and only if it is unpredictable.

In conclusion, in order to have a stream cipher that is confidentially safe against eavesdropping attacks and easy-to-use in real-world scenarios, we need to implement an unpredictable PRG and employ it in a Vernam cipher.

Appendix B

Genetic Programming

Contents

Genetic Programming (GP) is a type of Evolutionary Algorithm (EA) that automatically creates and evolves computer programs to solve a given problem [5]. It is inspired by the process of natural selection and genetics, using mechanisms similar to those found in biological evolution, such as selection, crossover (recombination), and mutation.

B.1 Initialization

The initialization of the population is a crucial step since the quality and diversity of the initial population can meaningfully impact the convergence of the population in the evolutionary run. Common initialization methods include: ramped grow, ramped full, and ramped half-and-half [313, 347]. Each method aims at creating a diverse initial population with varying tree structures and depths.

The ramped grow method initializes potentially unbalanced trees with varying depths and a mix of internal and terminal nodes at all levels. In Algorithm 3, we show a basic version of this algorithm.

Algorithm 3 Ramped Grow Initialization (`RampedGrow`)

Input
  $\mathcal{F}$ the function set
  $\mathcal{T}$ the terminal set
  d the depth of the tree
Output
  node a tree
if $d = 0$ or `RandomUniform`$(0, 1) < \frac{|\mathcal{T}|}{|\mathcal{T}|+|\mathcal{F}|}$ **then**
  return `SampleRandomNode`$(\mathcal{T})$
else
  node $\leftarrow$ `SampleRandomNode`$(\mathcal{F})$
  for each *child* of *node* **do**
    node.child $\leftarrow$ `RampedGrow`$(\mathcal{F}, \mathcal{T}, d-1)$
  end for
  return *node*
end if

The ramped full method generates trees that are fully balanced up to a certain depth (Algorithm 4).

Algorithm 4 Ramped Full Initialization (`RampedFull`)

Input
  $\mathcal{F}$ the function set
  $\mathcal{T}$ the terminal set
  d the depth of the tree
Output
  node a tree
if $d = 0$ **then**
  return `SampleRandomNode`$(\mathcal{T})$
else
  $node \leftarrow$ `SampleRandomNode`$(\mathcal{F})$
  for each *child* of *node* **do**
    $node.child \leftarrow$ `RampedFull`$(\mathcal{F}, \mathcal{T}, d-1)$
  end for
  return *node*
end if

The ramped half-and-half method combines both the ramped grow and ramped full methods to create a diverse population with a mix of tree structures with different depths. The combination of the previous methods helps to balance diversity and depth, facilitating a more global exploration of the search space. In Algorithm 5, we show a randomized implementation of ramped half-and-half for generating a single GP tree.

Algorithm 5 Ramped Half-and-Half Initialization (`RampedHalfHalf`)

Input
  $\mathcal{F}$ the function set
  $\mathcal{T}$ the terminal set
  d_{min} the minimum depth of the tree
  d_{max} the maximum depth of the tree
Output
  node a tree
$d \leftarrow$ `RandomInteger`$(d_{\text{min}}, d_{\text{max}})$
if `RandomUniform`$(0, 1) < \frac{1}{2}$ **then**
  return `RampedGrow`$(\mathcal{F}, \mathcal{T}, d)$
else
  return `RampedFull`$(\mathcal{F}, \mathcal{T}, d)$
end if

B.2 Evaluation and Selection

After the population P is initialized, the main evolutionary cycle, consisting of several generations, can begin. In the first step of the evolutionary cycle, current solutions are evaluated according to the provided fitness function f. After the evaluation has been performed, a selection algorithm is applied to the population to

select the individuals that are joining the mating pool. For the sake of completeness, we are going to describe the main selection algorithms that can be employed. We remark that these selection algorithms are not specific for GP. In the following pseudo-codes, we assume that we are dealing with a single-objective fitness function that we want to maximize.

Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population, where the winner of each tournament is selected for joining the mating pool (Algorithm 6).

Algorithm 6 Tournament Selection (`TournamentSel`)

Input
  P the population
  t the tournament size
Output
  best the best tree in the tournament
best $\leftarrow$ `SampleRandomIndividual`(P)
for $i \leftarrow 1$ to $t - 1$ **do**
  competitor $\leftarrow$ `SampleRandomIndividual`(P)
  if *competitor.fitness* $>$ *best.fitness* **then**
    best $\leftarrow$ *competitor*
  end if
end for
return *best*

Roulette wheel selection, also known as fitness proportionate selection, selects individuals based on their fitness, where individuals with better fitness have a higher probability of being selected (Algorithm 7).

Algorithm 7 Roulette Wheel Selection (`RouletteSel`)

Input
  P the population
Output
  tree the sampled tree
$s \leftarrow \sum_{i=1}^{|P|} P[i].fitness$
$p \leftarrow [\,]$
for $i \leftarrow 1$ to $|P|$ **do**
  `Append`$(p, \frac{P[i].fitness}{s})$
end for
return `WeightedSampleRandomIndividual`(P, p)

Rank-based selection assigns a selection probability to individuals based on their rank rather than their fitness (Algorithm 8).

In case of optimization problems in which the individual needs to be evaluated on a sequence of test cases $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in X, \mathbf{y}_i \in Y, i \in [1..N]\}$ with $|D| = N$, where X denotes the input space and Y denotes the output space, to compute the fitness value, the lexicase selection can be employed [303, 302, 305, 304].

Lexicase selection considers each individual on a case-by-case basis across all test cases. It allows for individuals that perform well on different subsets of test cases

Algorithm 8 Rank-based Selection (`RankSel`)

Input
    P the population
Output
    tree the sampled tree
$P \leftarrow$ `SortAscending`$(P,$ "fitness"$)$
for $i \leftarrow 1$ to $|P|$ **do**
    $P[i].rank \leftarrow i$
end for
$s \leftarrow \sum_{i=1}^{|P|} P[i].rank$
$p \leftarrow [\,]$
for $i \leftarrow 1$ to $|P|$ **do**
    `Append`$(p, \frac{P[i].rank}{s})$
end for
return `WeightedSampleRandomIndividual`(P, p)

to be selected (Algorithm 9).

Algorithm 9 Lexicase Selection (`LexicaseSel`)

Input
    P the population
    D the set of test cases
Output
    best the best tree in the selection
$\tilde{P} \leftarrow P$
$\tilde{D} \leftarrow$ `RandomShuffle`(D)
for $i \leftarrow 1$ to $|\tilde{D}|$ **do**
    $v \leftarrow \max_{s \in \tilde{P}} s.fitness[\tilde{D}[i]]$
    $\tilde{P} \leftarrow \{s \in \tilde{P} \mid s.fitness[\tilde{D}[i]] = v\}$
    if $|\tilde{P}| = 1$ **then**
        return $\tilde{P}[1]$
    end if
end for
return `SampleRandomIndividual`$(\tilde{P})$

A down-sampled variant of lexicase selection and, in general, selection methods involving a sequence of test cases can be adopted to reduce the computational effort and improve the generalization capabilities of the evolved solutions when evaluated on different test cases [300, 299, 301]. In this variant, at the beginning of each generation, before the actual selection algorithm takes place, a fixed-size subset of the available test cases is sampled without replacement and utilized to perform the selection.

B.3 Recombination and Mutation

The crossover and mutation operators are defined to operate on tree-based representations of candidate solutions for the given optimization problem. Crossover and mutation are the primary genetic operators that modify the structure of individuals

in the population. These operators are crucial for introducing diversity and enabling both the exploration and exploitation of the search space. Both crossover and mutation are usually associated with an execution probability, which states the likelihood that the operator is performed. Common tree-based recombination and mutation operators are sub-tree crossover and sub-tree mutation, whose pseudo-codes are following described. In these pseudo-codes, the function `ReplaceSubTreeWith` gets a tree as first parameter, a sub-tree of the given tree as second parameter, and another tree as third parameter. It replaces the sub-tree given as second parameter in the tree given as first parameter with the tree given as third parameter.

Crossover involves exchanging sub-trees between two parent individuals to produce offspring (Algorithm 10).

Algorithm 10 Sub-tree Crossover (`SubTreeCx`)

Input
  s_1 the first parent individual
  s_2 the second parent individual
Output
  c_1 the first child individual
  c_2 the second child individual
$i_1 \leftarrow$ `SelectRandomNode`(s_1)
$i_2 \leftarrow$ `SelectRandomNode`(s_2)
$t_1 \leftarrow$ `ExtractSubTreeRootedAtNode`(i_1)
$t_2 \leftarrow$ `ExtractSubTreeRootedAtNode`(i_2)
$c_1 \leftarrow$ `DeepCopy`(s_1)
$c_2 \leftarrow$ `DeepCopy`(s_2)
$c_1 \leftarrow$ `ReplaceSubTreeWith`(c_1, t_1, t_2)
$c_2 \leftarrow$ `ReplaceSubTreeWith`(c_2, t_2, t_1)
return c_1, c_2

Mutation involves making random modifications to an individual structure, typically by altering nodes or sub-trees (Algorithm 11).

Algorithm 11 Sub-tree Mutation (`SubTreeMut`)

Input
  s the input individual
  $\mathcal{F}$ the function set
  $\mathcal{T}$ the terminal set
  $d_{\min}$ the minimum depth of the tree
  $d_{\max}$ the maximum depth of the tree
Output
  c the mutated individual
$i \leftarrow$ `SelectRandomNode`(s)
$t \leftarrow$ `ExtractSubTreeRootedAtNode`(i)
$c \leftarrow$ `DeepCopy`(s)
▷ Other initialization methods could be adopted
$\tilde{t} \leftarrow$ `RampedHalfHalf`$(\mathcal{F}, \mathcal{T}, d_{\min}, d_{\max})$
$c \leftarrow$ `ReplaceSubTreeWith`$(c, t, \tilde{t})$
return c

Appendix C

Deep Neural Networks

Contents

Deep Learning (DL) is a subset of Machine Learning (ML), i.e., a technique that enables computer systems to improve with data and experience, which in turn is a subset of Artificial Intelligence (AI). It involves the use of neural networks with many layers (hence "deep") to model complex patterns in data.

An Artificial Neural Network (ANN) is a computational model that consists of layers of interconnected nodes (neurons). Its development was inspired by the functioning of the brain. The general goal of a neural network consists in transforming and mapping numerical input data into a more compact representation by incrementally and automatically extracting macro-features from the input data and the intermediate results. The idea is to provide a way to model complex non-linear relationships by introducing representations that are expressed in terms of other, possibly simpler, representations.

Neural networks are commonly employed for classification and regression tasks. Moreover, they are utilized to compress high-dimensional and complex data into low-dimensional data, so that they can be better handled by ML algorithms.

C.1 Feed-Forward Architectures

The most popular neural network is the feed-forward neural network, also called Multi-Layer Perceptron (MLP), where connections do not form cycles and the network is modeled as a directed acyclic graph. The most basic and early type of neural network is the perceptron [348], which takes n inputs and produces a single output. Mathematically, it can be represented as:

$$y = \mathbf{1}(\mathbf{W} \cdot \mathbf{x} + b)$$

where:

- $\mathbf{x} \in \mathbb{R}^n$ is the input vector;
- $\mathbf{W} \in \mathbb{R}^n$ is the weight vector;
- $b \in \mathbb{R}$ is the bias term;

- $\mathbf{1}$ is the Heaviside step function, defined as:

$$\mathbf{1}(z) = \begin{cases} 0 & \text{if } z \leq 0, \\ 1 & \text{if } z > 0. \end{cases}$$

In this case, the Heaviside step function is employed as activation function, which is a core component in neural networks design. In practical cases, other activation functions are employed since the properties of the Heaviside step function do not enable a neural network to learn from data.

Activation functions introduce non-linearity into the network, allowing it to learn complex patterns. Common activation functions include:

- **Sigmoid**: the sigmoid function maps any real-valued number to the range (0, 1). It is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  The sigmoid function is often used in the output layer for binary classification tasks. However, it can suffer from vanishing gradients;

- **Hyperbolic Tangent (tanh)**: the tanh function maps any real-valued number to the range (-1, 1). It is defined as:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

  The tanh function is zero-centered, which can make optimization easier. However, it can also suffer from vanishing gradients;

- **Rectified Linear Unit (ReLU)**: the ReLU function is defined as:

$$\text{ReLU}(z) = \max(0, z)$$

  ReLU is widely used in hidden layers due to its simplicity and efficiency. It mitigates the vanishing gradient problem but can suffer from the "dying ReLU" problem, where neurons can become inactive;

- **Leaky ReLU**: the Leaky ReLU function is a variant of ReLU that allows a small, non-zero gradient when the unit is inactive. It is defined as:

$$\text{LeakyReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{otherwise} \end{cases}$$

  where α is a small constant (e.g., 0.01). Leaky ReLU helps to mitigate the dying ReLU problem by allowing a small gradient when the input is negative.

We are going to denote a generic activation function as $g : \mathbb{R} \to \mathbb{R}$. Each of the described activation functions can be utilized in place of the Heaviside step function in the aforementioned perceptron-based neural network architecture to enable the network to learn (i.e., tune weights and bias) from input data. Moreover, additional outputs can be added to model more complex problems. By putting all together, we can mathematically formulate a neural network with an arbitrary activation function and no hidden layers (the output layer is the only layer of the network)

that takes n real-valued numbers as input and produces m real-valued numbers as output:

$$\mathbf{y} = g(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

where:

- $\mathbf{x} \in \mathbb{R}^n$ is the input vector;
- $\mathbf{y} \in \mathbb{R}^m$ is the output vector;
- $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix;
- $\mathbf{b} \in \mathbb{R}^m$ is the bias vector;
- g is the activation function.

A shallow neural network consists of one hidden layer. Mathematically, it can be represented as:

$$\mathbf{h} = g_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{y} = g_2(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

where:

- $\mathbf{x} \in \mathbb{R}^n$ is the input vector;
- $\mathbf{y} \in \mathbb{R}^k$ is the output vector;
- $\mathbf{W}_1 \in \mathbb{R}^{m \times n}$ is the weight matrix for the hidden layer;
- $\mathbf{b}_1 \in \mathbb{R}^m$ is the bias vector for the hidden layer;
- $\mathbf{W}_2 \in \mathbb{R}^{k \times m}$ is the weight matrix for the output layer;
- $\mathbf{b}_2 \in \mathbb{R}^k$ is the bias vector for the output layer;
- g_1 and g_2 are activation functions for the hidden and output layers, respectively.

A Deep Neural Network (DNN) extends the concept of a shallow neural network by adding more hidden layers. The presence of multiple hidden layers enables the network to approximate increasingly complex functions. Mathematically, it can be represented as:

$$\mathbf{h}_1 = g_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = g_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\vdots$$
$$\mathbf{y} = g_L(\mathbf{W}_L\mathbf{h}_{L-1} + \mathbf{b}_L)$$

where L is the total number of layers, that is, the number of hidden layers plus one (the output layer is a layer itself).

C.2 Learning

Training a neural network means tuning the weights and biases of the network to make the model learn to solve a given task. A classical example consists in teaching a neural network to recognize which hand-written digit is depicted in a given input image containing a hand-written digit. Before delving into how a neural network is trained, we need to provide an overview of the basic gradient-based optimization algorithms that can be adopted to solve continuous optimization problems.

C.2.1 Gradient-based Optimization

Let $J : \mathbb{R}^d \to \mathbb{R}$ be a function and let $\mathbf{w} \in \mathbb{R}^d$ be a generic vector of coefficients that are fed as input to J. A standard optimization problem consists in finding $\mathbf{w}^*$ such that:

$$\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^d} J(\mathbf{w})$$

The complexity of this problem depends upon the properties of J. As a matter of fact, the provided formulation asks to find a global minimum of J. In case J is not a convex function, then potentially many local minima with different values exist along with saddle points and flat regions. This means that, in general, we do not have the guarantee to find the optimal solution $\mathbf{w}^*$.

Non-convex optimization problems are at least NP-Hard. Many existing problems can be modeled as non-convex optimization ones. Therefore, algorithms that are capable of discovering good solutions, although not optimal, for non-convex problems depict great interest in the scientific community.

In case J is a differentiable function, gradient descent algorithm can be employed to tune the coefficients $\mathbf{w}$ to minimize J as best as possible. Since, in general, J is non-convex, gradient descent may provide a solution that does not correspond to the optimal one, depending on many factors such as coefficients initialization and shape of J.

Firstly, coefficients in $\mathbf{w}$ are initialized. The way initialization is performed impacts on the final solution discovered after the algorithm is terminated. If the initialization leads to initial coefficients that imply a value of J that is close to a bad local minimum, then it is likely that gradient descent remains stuck in that region without reaching possibly better local minima. A trivial way to initialize $\mathbf{w}$ consists in initializing the coefficients with either zero or a random value sampled by, for instance, a normal distribution.

After the initialization is performed, the actual iterations of gradient descent begin. In a generic iteration, $J(\mathbf{w})$ is computed along with the gradient ∇J:

$$\nabla J = \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \dots, \frac{\partial J}{\partial w_d} \right)^\top$$

The coefficients are updated according to the following update rule:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J$$

where $\eta > 0$ is the learning rate. η represents the amount that the coefficients are updated during optimization and it is usually a small value in $(0, 1)$. It controls the speed at which J is updated. Generally, a large η allows the optimization to run faster, at the cost of reaching a sub-optimal vector of coefficients. A smaller η may allow the optimization to reach a more optimal or even globally optimal vector of coefficients but may take significantly longer.

The iterations are sequentially executed until a termination criterion is met (e.g., number of iterations, convergence of J, execution time).

In a typical DL context, we have a DNN, which is basically a function $f_{\mathbf{w}} : \mathbb{R}^n \to \mathbb{R}^k$ that depends upon a vector of coefficients $\mathbf{w} \in \mathbb{R}^d$ containing all the weights and biases of the network, and a training set $D_{\text{train}} = \{(\mathbf{x}_i, \mathbf{y}_i) \mid \mathbf{x}_i \in \mathbb{R}^n, \mathbf{y}_i \in \mathbb{R}^k, i \in [1..N]\}$ with $|D_{\text{train}}| = N$, where $\mathbf{x}_i$ is a generic data point and $\mathbf{y}_i$ is the corresponding ground-truth. The task consists in finding $\mathbf{w}$ such that $f_{\mathbf{w}}$ is able to provide a good approximation of D_{train}. The ultimate goal consists in using $f_{\mathbf{w}}$ for unseen data points, which, in general, do not have a ground-truth and thus it needs to be estimated.

Provided that $f_{\mathbf{w}}$ employs differentiable activation functions, we can model J as a loss function in which a measure of distance between the expected output and the generated output is computed and then we can use gradient descent to optimize the weights and biases of the network so that the loss is minimized and the training set distribution is learned.

A common choice is the Mean Squared Error (MSE) loss function, which is defined as:

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \frac{\|f_{\mathbf{w}}(\mathbf{x}_i) - \mathbf{y}_i\|^2}{2}$$

The MSE loss function $J(\mathbf{w})$ quantifies the average squared difference between the predicted output and the actual ground truth over all training examples.

In this standard version, gradient descent can be employed but, in practical cases, when N is large, it can take a long time since the gradients for all training examples must be separately calculated and then averaged.

To overcome this issue, Stochastic Gradient Descent (SGD) is employed. In this variant, the optimization consists of a series of epochs. In a generic epoch, the data points in the training set are randomly shuffled. Then, given a batch size $m > 0$ with N multiple of m, the shuffled training set is partitioned in $\frac{N}{m}$ blocks of data points called batches. Batches are iterated sequentially and each batch is employed as a set of data points for performing a single vanilla gradient descent iteration, i.e., a single weights and biases update.

The MSE is an example of loss function that can be written as an average:

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} J_i(\mathbf{w}) \quad \text{where} \quad J_i(\mathbf{w}) = \frac{\|f_{\mathbf{w}}(\mathbf{x}_i) - \mathbf{y}_i\|^2}{2}$$

Provided that, let j_1 be the index of the first training example in the current batch

and let $j_2 = j_1 + m - 1$ be the index of the last training example in the current batch. The loss function for this batch is formulated as:

$$J_{(j_1,j_2)}(\mathbf{w}) = \frac{1}{j_2 - j_1 + 1} \sum_{i=j_1}^{j_2} J_i(\mathbf{w})$$

and the update rule, consequently, is:

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{j_2 - j_1 + 1} \sum_{i=j_1}^{j_2} \nabla J_i$$

The larger the batch size is, the better is the approximation with a gradient descent that uses all the training examples at once to compute the gradients and the slower is the computation. When the batch size is one, we say that we are training the neural network in an online fashion, which is useful when N is not known in advance (e.g., when data are generated in a real-time stream).

C.2.2 Backpropagation Algorithm

In the context of neural networks training, an algorithm such as SGD is said to be an "optimizer", that is, a strategy to update weights and biases of the network w.r.t. the error to make the network learn from data. The optimizer is employed as a main component in the actual learning algorithm utilized to train a neural network. This learning algorithm is called backpropagation [349].

The backpropagation algorithm works by propagating the error of the network's output backward through the layers, allowing the network to adjust its weights and biases to minimize this error. This algorithm is an application of the chain rule of calculus and it is employed in conjunction with SGD (or its variants) to update the network parameters.

Consider a DNN with L layers, where each layer l with $l \in [1..L]$ has an associated weight matrix $\mathbf{W}_l$, a bias vector $\mathbf{b}_l$, and an activation function g_l. The i-th input to the network is denoted by $\mathbf{x}_i$, and the corresponding predicted output by $\tilde{\mathbf{y}}_i$. The goal is to minimize a loss function $J(\mathbf{w})$, with $\mathbf{w}$ containing all $\mathbf{W}_l$ and $\mathbf{b}_l$, such as the MSE, by updating the weights $\mathbf{W}_l$ and biases $\mathbf{b}_l$ in the network. In particular, $\mathbf{b}_l$ is a real-valued vector whose i-th component corresponds to the bias of the i-th neuron at layer l, while $\mathbf{W}_l$ is a real-valued matrix such that the component at the i-th row and j-th column corresponds to the weight located on the edge connecting the j-th neuron at layer $l-1$ with the i-th neuron at layer l.

The loss function must satisfy two assumptions:

- It can be written as an average:

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} J_i(\mathbf{w})$$

  where $J_i(\mathbf{w})$ is the loss function computed on the i-th training example;

- It can be written as a function of the outputs from the neural network.

The backpropagation algorithm is divided in forward pass and backward pass. In the forward pass, the input $\mathbf{x}_i$ is propagated through the network layer by layer to produce the output $\tilde{\mathbf{y}}_i$. For each layer l, the computation is as follows:

$$\mathbf{h}_{l,i} = g_l(\mathbf{z}_{l,i}) = g_l(\mathbf{W}_l\mathbf{h}_{l-1,i} + \mathbf{b}_l)$$

where:

- $\mathbf{h}_{0,i} = \mathbf{x}_i$ is the i-th input to the network;
- $\mathbf{z}_{l,i} = \mathbf{W}_l\mathbf{h}_{l-1,i} + \mathbf{b}_l$ is the pre-activation value at layer l;
- g_l is the activation function for layer l;
- $\mathbf{h}_{l,i}$ is the output of layer l.

The output of the network after the last layer L is given by $\tilde{\mathbf{y}}_i = \mathbf{h}_{L,i}$.

After obtaining the output, the error between the predicted output $\tilde{\mathbf{y}}_i$ and the actual target $\mathbf{y}_i$ is computed by using the loss function $J_i(\mathbf{w})$.

In the backward pass, the algorithm updates the network parameters by propagating this error backward through the network, from the output layer to the input. This is done by computing the gradient of the loss function w.r.t. each parameter in the network (i.e., each $\mathbf{W}_l$ and $\mathbf{b}_l$) and adjusting them accordingly using the optimizer. The derivative of the error w.r.t. each parameter indicates how much a change in the parameter would change the error.

Given that $\odot$ is the Hadamard product (i.e., the element-wise multiplication), the gradient of the loss computed on the i-th training example w.r.t. the output layer pre-activation values $\mathbf{z}_{L,i}$ is:

$$\delta_{L,i} = \nabla_{\mathbf{z}_{L,i}} J_i = \nabla_{\mathbf{h}_{L,i}} J_i \odot g'_L(\mathbf{z}_{L,i})$$

where:

- $\nabla_{\mathbf{h}_{L,i}} J_i$ is the gradient of the loss computed on the i-th training example w.r.t. the output $\mathbf{h}_{L,i}$;
- g'_L is the derivative of the activation function at the output layer.

For each layer l from $L-1$ down to one, we can compute the error gradient w.r.t. the pre-activation values $\mathbf{z}_{l,i}$:

$$\delta_{l,i} = (\mathbf{W}_{l+1}^\top \delta_{l+1,i}) \odot g'_l(\mathbf{z}_{l,i})$$

Here, $\delta_{l,i}$ represents the error signal at layer l for the i-th training example. In particular, we have that $\nabla_{\mathbf{W}_l} J_i = \delta_{l,i}\mathbf{h}_{l-1,i}^\top$ and $\nabla_{\mathbf{b}_l} J_i = \delta_{l,i}$.

This procedure computes the gradients for a single training example. By applying this procedure for several training inputs, we obtain several gradients that can be averaged over the training inputs to calculate a gradient that can be employed to update the weights and biases according to the given batch of examples.

A pseudo-code that summarizes the backpropagation algorithm with SGD as optimizer and a batch size of m is described in Algorithm 12. We assume that the number of training examples is a multiple of m.

Algorithm 12 Backpropagation Algorithm (`Backpropagation`)

Input
  D_{train} the training set
  m the batch size
  η the learning rate
  L the number of layers
  $\mathbf{W}_l$ the weight matrices for each layer l
  $\mathbf{b}_l$ the bias vectors for each layer l
  g_l the activation functions for each layer l
  J the loss function
Output
  $\mathbf{w}$ the final tuned weights and biases of the network
$N \leftarrow |D_{\text{train}}|$
while termination criterion not met **do**
  `InPlaceRandomShuffle`(D_{train})
  for $j \leftarrow 1$ to $\frac{N}{m}$ **do**
    $j_1 \leftarrow m(j-1)+1$
    $j_2 \leftarrow j_1 + m - 1$
    for $i \leftarrow j_1$ to j_2 **do**
      Forward pass: compute activations
      $\mathbf{h}_{0,i} \leftarrow \mathbf{x}_i$
      for $l \leftarrow 1$ to L **do**
        $\mathbf{z}_{l,i} \leftarrow \mathbf{W}_l \mathbf{h}_{l-1,i} + \mathbf{b}_l$
        $\mathbf{h}_{l,i} \leftarrow g_l(\mathbf{z}_{l,i})$
      end for
      Backward pass: compute gradients
      $\delta_{L,i} \leftarrow \nabla_{\mathbf{h}_{L,i}} J_i \odot g'_L(\mathbf{z}_{L,i})$
      for $l \leftarrow L-1$ down to 1 **do**
        $\delta_{l,i} \leftarrow (\mathbf{W}_{l+1}^\top \delta_{l+1,i}) \odot g'_l(\mathbf{z}_{l,i})$
      end for
    end for
    Update weights and biases
    for $l \leftarrow L$ down to 1 **do**
      $\mathbf{W}_l \leftarrow \mathbf{W}_l - \frac{\eta}{m} \sum_{i=j_1}^{j_2} \delta_{l,i} \mathbf{h}_{l-1,i}^\top$
      $\mathbf{b}_l \leftarrow \mathbf{b}_l - \frac{\eta}{m} \sum_{i=j_1}^{j_2} \delta_{l,i}$
    end for
  end for
end while
return $(\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L, \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_L)^\top$

Regularization

Regularization techniques can be employed during training to reduce overfitting (i.e., when the network nearly perfectly learns the training data in a way that it cannot generalize to unseen data).

L2 regularization adds a penalty equal to the sum of the squared values of the weights to the loss function (here, the vector of parameters does not include biases):

$$J_{\text{reg}}(\mathbf{w}) = J(\mathbf{w}) + \frac{\lambda}{2} \sum_i w_i^2$$

When $\lambda > 0$ (i.e., the regularization parameter) is small we prefer to minimize the original function. When λ is large we prefer to have small weights.

Dropout is another regularization technique that is applied at specific chosen layers in the network, excluding the output layer. In a given layer and during a given forward pass, dropout involves randomly setting a fraction of neurons to zero during training. This prevents neurons from co-adapting too much.

References

[1] Zachary C Lipton. "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery." In: *Queue* 16.3 (2018), pp. 31–57. DOI: 10.1145/3236386.3241340.

[2] Anna Jobin, Marcello Ienca, and Effy Vayena. "The global landscape of AI ethics guidelines". In: *Nature machine intelligence* 1.9 (2019), pp. 389–399. DOI: 10.1038/s42256-019-0088-2.

[3] Cynthia Rudin. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature machine intelligence* 1.5 (2019), pp. 206–215. DOI: 10.1038/s42256-019-0048-x.

[4] John H Holland. "Genetic algorithms and the optimal allocation of trials". In: *SIAM journal on computing* 2.2 (1973), pp. 88–105. DOI: 10.1137/0202009.

[5] John R Koza. "Genetic programming as a means for programming computers by natural selection". In: *Statistics and computing* 4.2 (1994), pp. 87–112. DOI: 10.1007/BF00175355.

[6] Hans-Paul Paul Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. USA: John Wiley & Sons, Inc., 1993. ISBN: 0471571482. DOI: 10.5555/529401.

[7] Thomas Bäck and Hans-Paul Schwefel. "An Overview of Evolutionary Algorithms for Parameter Optimization". In: *Evolutionary Computation* 1.1 (1993), pp. 1–23. DOI: 10.1162/evco.1993.1.1.1.

[8] Hans-Paul Schwefel and Günter Rudolph. "Contemporary evolution strategies". In: *European conference on artificial life*. Springer. 1995, pp. 891–907. DOI: 10.1007/3-540-59496-5_351.

[9] R. Storn. "On the usage of differential evolution for function optimization". In: *Proceedings of North American Fuzzy Information Processing*. 1996, pp. 519–523. DOI: 10.1109/NAFIPS.1996.534789.

[10] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[11] Rainer Storn and Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of global optimization* 11 (1997), pp. 341–359. DOI: 10.1023/a:1008202821328.

[12] David B. Fogel. "Artificial Intelligence through Simulated Evolution". In: *Evolutionary Computation: The Fossil Record*. 1998, pp. 227–296. DOI: 10.1109/9780470544600.ch7.

[13] Lawrence J. Fogel. *Intelligence through simulated evolution: forty years of evolutionary programming*. USA: John Wiley & Sons, Inc., 1999. ISBN: 047133250X. DOI: 10.5555/317034.

[14] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. Springer, 2015. DOI: 10.1007/978-3-662-44874-8.

[15] Michael O'Neill and Conor Ryan. "Grammatical evolution". In: *IEEE Transactions on Evolutionary Computation* 5.4 (2001), pp. 349–358. DOI: 10.1109/4235.942529.

[16] John McCarthy. "History of LISP". In: *History of programming languages*. 1978, pp. 173–185. DOI: 10.1145/800025.1198360.

[17] Markus F Brameier and Wolfgang Banzhaf. *Basic concepts of linear genetic programming*. Springer, 2007. DOI: 10.1007/978-0-387-31030-5_2.

[18] Julian Miller and Andrew Turner. "Cartesian Genetic Programming". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO Companion '15. Madrid, Spain: Association for Computing Machinery, 2015, pp. 179–198. ISBN: 9781450334884. DOI: 10.1145/2739482.2756571.

[19] Julian F. Miller and Peter Thomson. "Cartesian Genetic Programming". In: *Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 121–132. ISBN: 978-3-540-46239-2.

[20] Alex A Freitas. "Comprehensible classification models: a position paper". In: *ACM SIGKDD explorations newsletter* 15.1 (2014), pp. 1–10. DOI: 10.1145/2594473.2594475.

[21] J Scott Armstrong and Fred Collopy. "Error measures for generalizing about forecasting methods: Empirical comparisons". In: *International journal of forecasting* 8.1 (1992), pp. 69–80. DOI: 10.1016/0169-2070(92)90008-W.

[22] Jan G. De Gooijer and Rob J. Hyndman. "25 years of time series forecasting". In: *International Journal of Forecasting* 22.3 (2006). Twenty five years of forecasting, pp. 443–473. ISSN: 0169-2070. DOI: 10.1016/j.ijforecast.2006.01.001.

[23] Lior Rokach and Oded Maimon. "Clustering Methods". In: *Data Mining and Knowledge Discovery Handbook*. Ed. by Oded Maimon and Lior Rokach. Boston, MA: Springer US, 2005, pp. 321–352. ISBN: 978-0-387-25465-4. DOI: 10.1007/0-387-25465-X_15.

[24] Weikuan Jia, Meili Sun, Jian Lian, and Sujuan Hou. "Feature dimensionality reduction: a review". In: *Complex & Intelligent Systems* 8.3 (2022), pp. 2663–2693. DOI: 10.1007/s40747-021-00637-x.

[25] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM Comput. Surv.* 41.3 (July 2009). ISSN: 0360-0300. DOI: 10.1145/1541880.1541882.

[26] Johannes Fürnkranz and Tomáš Kliegr. "A brief overview of rule learning". In: *International symposium on rules and rule markup languages for the semantic web*. Springer. 2015, pp. 54–69. DOI: 10.1007/978-3-319-21542-6_4.

[27] John R Koza. "Survey of genetic algorithms and genetic programming". In: *Wescon conference record*. Western Periodicals Company. 1995, pp. 589–594.

[28] Kenneth E Kinnear and Peter J Angeline. *Advances in genetic programming*. Vol. 3. MIT press Cambridge, MA, 1994.

[29] W. Banzhaf, J.R. Koza, C. Ryan, L. Spector, and C. Jacob. "Genetic programming". In: *IEEE Intelligent Systems and their Applications* 15.3 (2000), pp. 74–84. DOI: 10.1109/5254.846288.

[30] Milad Taleby Ahvanooey, Qianmu Li, Ming Wu, and Shuo Wang. "A survey of genetic programming and its applications". In: *KSII Transactions on Internet and Information Systems (TIIS)* 13.4 (2019), pp. 1765–1794.

[31] Vipul K Dabhi and Sanjay Chaudhary. "Empirical modeling using genetic programming: A survey of issues and approaches". In: *Natural Computing* 14 (2015), pp. 303–330. DOI: 10.1007/s11047-014-9416-y.

[32] Pedro G. Espejo, Sebastián Ventura, and Francisco Herrera. "A Survey on the Application of Genetic Programming to Classification". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.2 (2010), pp. 121–144. DOI: 10.1109/TSMCC.2009.2033566.

[33] David R White, James McDermott, Mauro Castelli, Luca Manzoni, Brian W Goldman, Gabriel Kronberger, Wojciech Jaśkowski, Una-May O'Reilly, and Sean Luke. "Better GP benchmarks: community survey results and proposals". In: *Genetic Programming and Evolvable Machines* 14 (2013), pp. 3–29. DOI: 10.1007/s10710-012-9177-2.

[34] Quang Huynh, Hemant Singh, Tapabrata Ray, and Akira Oyama. "Improved Genetic Programming for Symbolic Regression: Case Studies on Practical Applications". In: *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2022, pp. 1135–1142. DOI: 10.1109/SSCI51031.2022.10022279.

[35] Patryk Orzechowski, William La Cava, and Jason H. Moore. "Where are we now? a large benchmark study of recent symbolic regression methods". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, pp. 1183–1190. ISBN: 9781450356183. DOI: 10.1145/3205455.3205539.

[36] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio Olivetti de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H Moore. "Contemporary Symbolic Regression Methods and their Relative Performance". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2021. DOI: PMC11074949.

[37] Marco Virgolin and Solon P. Pissis. *Symbolic Regression is NP-hard*. 2022. DOI: 10.48550/arXiv.2207.01018. arXiv: 2207.01018 [cs.NE].

[38] Michael D Schmidt, Ravishankar R Vallabhajosyula, Jerry W Jenkins, Jonathan E Hood, Abhishek S Soni, John P Wikswo, and Hod Lipson. "Automated refinement and inference of analytical models for metabolic networks". In: *Physical biology* 8.5 (2011), p. 055011. DOI: 10.1088/1478-3975/8/5/055011.

[39] Michael Schmidt and Hod Lipson. "Distilling free-form natural laws from experimental data". In: *science* 324.5923 (2009), pp. 81–85. DOI: 10.1126/science.1165893.

[40] Josh C Bongard and Hod Lipson. "Nonlinear system identification using coevolution of models and tests". In: *IEEE Transactions on Evolutionary Computation* 9.4 (2005), pp. 361–384. DOI: 10.1109/TEVC.2005.850293.

[41] William La Cava, Kourosh Danai, Lee Spector, Paul Fleming, Alan Wright, and Matthew Lackner. "Automatic identification of wind turbine models using evolutionary multiobjective optimization". In: *Renewable Energy* 87 (2016), pp. 892–902. DOI: 10.1016/j.renene.2015.09.068.

[42] Shu-Heng Chen. *Genetic algorithms and genetic programming in computational finance*. Springer Science & Business Media, 2012. DOI: 10.1007/978-1-4615-0835-9_1.

[43] William La Cava, Kourosh Danai, and Lee Spector. "Inference of compact nonlinear dynamic models by epigenetic local search". In: *Engineering Applications of Artificial Intelligence* 55 (2016), pp. 292–306. DOI: 10.1016/j.engappai.2016.07.004.

[44] Karolina Stanislawska, Krzysztof Krawiec, and Zbigniew W Kundzewicz. "Modeling global temperature changes with genetic programming". In: *Computers & Mathematics with Applications* 64.12 (2012), pp. 3717–3728. DOI: 10.1016/j.camwa.2012.02.049.

[45] Yi Mei, Qi Chen, Andrew Lensen, Bing Xue, and Mengjie Zhang. "Explainable Artificial Intelligence by Genetic Programming: A Survey". In: *IEEE Transactions on Evolutionary Computation* 27.3 (2023), pp. 621–641. DOI: 10.1109/TEVC.2022.3225509.

[46] Andrew Lensen, Bing Xue, and Mengjie Zhang. "Genetic Programming for Evolving a Front of Interpretable Models for Data Visualization". In: *IEEE Transactions on Cybernetics* 51.11 (2021), pp. 5468–5482. DOI: 10.1109/TCYB.2020.2970198.

[47] Leonardo Augusto Ferreira, Frederico Gadelha Guimarães, and Rodrigo Silva. "Applying Genetic Programming to Improve Interpretability in Machine Learning Models". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. 2020, pp. 1–8. DOI: 10.1109/CEC48606.2020.9185620.

[48] Ting Hu. "Genetic Programming for Interpretable and Explainable Machine Learning". In: *Genetic Programming Theory and Practice XIX*. Ed. by Leonardo Trujillo, Stephan M. Winkler, Sara Silva, and Wolfgang Banzhaf. Singapore: Springer Nature Singapore, 2023, pp. 81–90. ISBN: 978-981-19-8460-0. DOI: 10.1007/978-981-19-8460-0_4.

[49] Giorgia Nadizar, Luigi Rovito, Andrea De Lorenzo, Eric Medvet, and Marco Virgolin. "An Analysis of the Ingredients for Learning Interpretable Symbolic Regression Models with Human-in-the-loop and Genetic Programming". In: *ACM Trans. Evol. Learn. Optim.* 4.1 (Feb. 2024). DOI: 10.1145/3643688.

[50] Marco Virgolin, Andrea De Lorenzo, Francesca Randone, Eric Medvet, and Mattias Wahde. "Model learning with personalized interpretability estimation (ML-PIE)". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '21. Lille, France: Association for Computing Machinery, 2021, pp. 1355–1364. ISBN: 9781450383516. DOI: 10.1145/3449726.3463166.

[51] Marco Virgolin, Andrea De Lorenzo, Eric Medvet, and Francesca Randone. "Learning a Formula of Interpretability to Learn Interpretable Formulas". In: *Parallel Problem Solving from Nature – PPSN XVI*. Ed. by Thomas Bäck, Mike Preuss, André Deutz, Hao Wang, Carola Doerr, Michael Emmerich, and Heike Trautmann. Cham: Springer International Publishing, 2020, pp. 79–93. ISBN: 978-3-030-58115-2. DOI: 10.1007/978-3-030-58115-2_6.

[52] Alberto Moraglio, Krzysztof Krawiec, and Colin Johnson. "Geometric Semantic Genetic Programming". In: vol. 7491. Sept. 2012, pp. 21–31. ISBN: 9783642329364. DOI: 10.1007/978-3-642-32937-1_3.

[53] Leonardo Vanneschi. "An introduction to geometric semantic genetic programming". In: *NEO 2015: Results of the Numerical and Evolutionary Optimization Workshop NEO 2015 held at September 23-25 2015 in Tijuana, Mexico*. Springer. 2016, pp. 3–42. DOI: 10.1007/978-3-319-44003-3_1.

[54] Mauro Castelli, Leonardo Vanneschi, and Sara Silva. "Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators". In: *Expert Systems with Applications* 40.17 (2013), pp. 6856–6862. DOI: 10.1016/j.eswa.2013.06.037.

[55] Mauro Castelli, Leonardo Vanneschi, and Sara Silva. "Prediction of the unified Parkinson's disease rating scale assessment using a genetic programming system with geometric semantic genetic operators". In: *Expert Systems with Applications* 41.10 (2014), pp. 4608–4616. DOI: 10.1016/j.eswa.2014.01.018.

[56] Leonardo Vanneschi, Sara Silva, Mauro Castelli, and Luca Manzoni. "Geometric semantic genetic programming for real life applications". In: *Genetic programming theory and practice xi* (2014), pp. 191–209. DOI: 10.1007/978-1-4939-0375-7_11.

[57] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. "A survey of semantic methods in genetic programming". In: *Genetic Programming and Evolvable Machines* 15 (2014), pp. 195–214. DOI: 10.1007/s10710-013-9210-0.

[58] Giorgia Nadizar, Fraser Garrow, Berfin Sakallioglu, Lorenzo Canonne, Sara Silva, and Leonardo Vanneschi. "An Investigation of Geometric Semantic GP with Linear Scaling". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 1165–1174. ISBN: 9798400701191. DOI: 10.1145/3583131.3590418.

[59] Giorgia Nadizar, Berfin Sakallioglu, Fraser Garrow, Sara Silva, and Leonardo Vanneschi. "Geometric semantic GP with linear scaling: Darwinian versus Lamarckian evolution". In: *Genetic Programming and Evolvable Machines* 25.2 (2024), pp. 1–24. DOI: 10.1007/s10710-024-09488-0.

[60] Gloria Pietropolli, Luca Manzoni, Alessia Paoletti, and Mauro Castelli. "Combining geometric semantic gp with gradient-descent optimization". In: *European Conference on Genetic Programming (Part of EvoStar)*. Springer. 2022, pp. 19–33. DOI: 10.1007/978-3-031-02056-8_2.

[61] Gloria Pietropolli, Luca Manzoni, Alessia Paoletti, and Mauro Castelli. "On the hybridization of geometric semantic GP with gradient-based optimizers". In: *Genetic Programming and Evolvable Machines* 24.2 (2023), p. 16. DOI: 10.1007/s10710-023-09463-1.

[62] Mauro Castelli, Luca Manzoni, Leonardo Vanneschi, Sara Silva, and Aleš Popovič. "Self-tuning geometric semantic genetic programming". In: *Genetic Programming and Evolvable Machines* 17 (2016), pp. 55–74. DOI: 10.1007/s10710-015-9251-7.

[63] Leonardo Vanneschi. "SLIM_GSGP: The non-bloating geometric semantic genetic programming". In: *European Conference on Genetic Programming (Part of EvoStar)*. Springer. 2024, pp. 125–141. DOI: 10.1007/978-3-031-56957-9_8.

[64] Mauro Castelli, Sara Silva, and Leonardo Vanneschi. "A C++ framework for geometric semantic genetic programming". In: *Genetic Programming and Evolvable Machines* 16 (2015), pp. 73–81. DOI: 10.1007/s10710-014-9218-0.

[65] Leonardo Vanneschi, Mauro Castelli, Luca Manzoni, and Sara Silva. "A new implementation of geometric semantic GP and its application to problems in pharmacokinetics". In: *Genetic Programming: 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013. Proceedings 16*. Springer. 2013, pp. 205–216. DOI: 10.1007/978-3-642-37207-0_18.

[66] Mauro Castelli, Leonardo Vanneschi, and Aleš Popovič. "Controlling individuals growth in semantic genetic programming through elitist replacement". In: *Computational intelligence and neuroscience* 2016 (2016), pp. 42–42. DOI: 10.1155/2016/8326760.

[67] Joao Francisco B. S. Martins, Luiz Otavio V. B. Oliveira, Luis F. Miranda, Felipe Casadei, and Gisele L. Pappa. "Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, pp. 1151–1158. ISBN: 9781450356183. DOI: 10.1145/3205455.3205593.

[68] Daik Koga and Kei Ohnishi. "Non-generational Geometric Semantic Genetic Programming". In: *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2021, pp. 1–7. DOI: 10.1109/SSCI50451.2021.9660116.

[69] Leonardo Trujillo, Jose Manuel Muñoz Contreras, Daniel E Hernandez, Mauro Castelli, and Juan J Tapia. "GSGP-CUDA—A CUDA framework for geometric semantic genetic programming". In: *SoftwareX* 18 (2022), p. 101085. DOI: 10.1016/j.softx.2022.101085.

[70] Tomasz P Pawlak, Bartosz Wieloch, and Krzysztof Krawiec. "Review and comparative analysis of geometric semantic crossovers". In: *Genetic Programming and Evolvable Machines* 16 (2015), pp. 351–386. DOI: 10.1007/s10710-014-9239-8.

[71] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[72] Claude Carlet, Yves Crama, and Peter L Hammer. *Boolean Functions for Cryptography and Error-Correcting Codes*. 2010.

[73] Claude Carlet. "Boolean Functions for Cryptography and Coding Theory". In: (2021).

[74] Oscar S Rothaus. "On "bent" functions". In: *Journal of Combinatorial Theory, Series A* 20.3 (1976), pp. 300–305. DOI: 10.1016/0097-3165(76)90024-8.

[75] Robert L McFarland. "A family of difference sets in non-cyclic groups". In: *Journal of Combinatorial Theory, Series A* 15.1 (1973), pp. 1–10. DOI: 10.1016/0097-3165(73)90031-9.

[76] Sihem Mesnager and Mesnager. *Bent functions*. Vol. 1. Springer, 2016. DOI: 10.1007/978-3-319-32595-8.

[77] Na Li, Longjiang Qu, Wen-Feng Qi, GuoZhu Feng, Chao Li, and DuanQiang Xie. "On the construction of Boolean functions with optimal algebraic immunity". In: *IEEE Transactions on Information Theory* 54.3 (2008), pp. 1330–1334. DOI: 10.1109/TIT.2007.915914.

[78] Yindong Chen and Peizhong Lu. "Two classes of symmetric Boolean functions with optimum algebraic immunity: Construction and analysis". In: *IEEE transactions on information theory* 57.4 (2011), pp. 2522–2538. DOI: 10.1109/TIT.2011.2111810.

[79] Ziran Tu and Yingpu Deng. "A conjecture about binary strings and its applications on constructing Boolean functions with optimal algebraic immunity". In: *Designs, Codes and Cryptography* 60.1 (2011), pp. 1–14. DOI: 10.1007/s10623-010-9413-9.

[80] Linda Burnett, William Millan, Edward Dawson, Andrew Clark, et al. "Simpler methods for generating better Boolean functions with good cryptographic properties". In: *Australasian Journal of Combinatorics* 29 (2004), pp. 231–248.

[81] John A Clark, Jeremy L Jacob, Subhamoy Maitra, and Pantelimon Stănică. "Almost Boolean functions: The design of Boolean functions by spectral inversion". In: *Computational Intelligence* 20.3 (2004), pp. 450–462. DOI: 10.1111/j.0824-7935.2004.00245.x.

[82] John A Clark, Jeremy L Jacob, Susan Stepney, Subhamoy Maitra, and William Millan. "Evolving Boolean functions satisfying multiple criteria". In: *International conference on cryptology in India*. Springer. 2002, pp. 246–259. DOI: 10.1007/3-540-36231-2_20.

[83] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[84] Isaac López-López, Guillermo Sosa-Gómez, Carlos Segura, Diego Oliva, and Omar Rojas. "Metaheuristics in the optimization of cryptographic Boolean functions". In: *Entropy* 22.9 (2020), p. 1052. DOI: 10.3390/e22091052.

[85] Karlo Knežević. "Combinatorial optimization in cryptography". In: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2017, pp. 1324–1330. DOI: 10.23919/MIPRO.2017.7973628.

[86] Hernán Aguirre, Hiroyuki Okazaki, and Yasushi Fuwa. "An evolutionary multiobjective approach to design highly non-linear Boolean functions". In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. GECCO '07. London, England: Association for Computing Machinery, 2007, pp. 749–756. ISBN: 9781595936974. DOI: 10.1145/1276958.1277112.

[87] William Millan, Andrew Clark, and Ed Dawson. "Heuristic design of cryptographically strong balanced Boolean functions". In: *International conference on the theory and applications of cryptographic techniques*. Springer. 1998, pp. 489–499. DOI: 10.1007/BFb0054148.

[88] William Millan, Andrew Clark, and Ed Dawson. "An effective genetic algorithm for finding highly nonlinear Boolean functions". In: *International conference on information and communications security*. Springer. 1997, pp. 149–158. DOI: 10.1007/BFb0028471.

[89] A Dimovski and D Gligoroski. "Generating highly nonlinear Boolean functions using a genetic algorithm". In: *6th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Service, 2003. TELSIKS 2003*. Vol. 2. IEEE. 2003, pp. 604–607. DOI: 10.1109/TELSKS.2003.1246297.

[90] Rajesh Asthana, Neelam Verma, and Ram Ratan. "Generation of Boolean functions using Genetic Algorithm for cryptographic applications". In: *2014 IEEE International Advance Computing Conference (IACC)*. IEEE. 2014, pp. 1361–1366. DOI: 10.1109/IAdCC.2014.6779525.

[91] Pratap Kumar Behera and Sugata Gangopadhyay. "An improved hybrid genetic algorithm to construct balanced Boolean function with optimal cryptographic properties". In: *Evolutionary Intelligence* 15.1 (2022), pp. 639–653. DOI: 10.1007/s12065-020-00538-x.

[92] Luca Mariot and Alberto Leporati. "A genetic algorithm for evolving plateaued cryptographic boolean functions". In: *International Conference on Theory and Practice of Natural Computing*. Springer. 2015, pp. 33–45. DOI: 10.1007/978-3-319-26841-5_3.

[93] Yuliang Zheng and Xian-Mo Zhang. "Plateaued functions". In: *International Conference on Information and Communications Security*. Springer. 1999, pp. 284–300. DOI: 10.1007/978-3-540-47942-0_24.

[94] Luca Mariot and Alberto Leporati. "Heuristic Search by Particle Swarm Optimization of Boolean Functions for Cryptographic Applications". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO Companion '15. Madrid, Spain: Association for Computing Machinery, 2015, pp. 1425–1426. ISBN: 9781450334884. DOI: 10.1145/2739482.2764674.

[95] Julian F Miller et al. "An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach". In: *Proceedings of the genetic and evolutionary computation conference*. Vol. 2. 1999, pp. 1135–1142.

[96] Stjepan Picek, Domagoj Jakobovic, Julian F Miller, Elena Marchiori, and Lejla Batina. "Evolutionary methods for the construction of cryptographic boolean functions". In: *European Conference on Genetic Programming*. Springer. 2015, pp. 192–204. DOI: 10.1007/978-3-319-16501-1_16.

[97] Stjepan Picek, Domagoj Jakobovic, and Marin Golub. "Evolving cryptographically sound boolean functions". In: *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '13 Companion. Amsterdam, The Netherlands: Association for Computing Machinery, 2013, pp. 191–192. ISBN: 9781450319645. DOI: 10.1145/2464576.2464671.

[98] Radek Hrbacek and Vaclav Dvorak. "Bent function synthesis by means of Cartesian genetic programming". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2014, pp. 414–423. DOI: 10.1007/978-3-319-10762-2_41.

[99] Jakub Husa and Roland Dobai. "Designing bent boolean functions with parallelized linear genetic programming". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '17. Berlin, Germany: Association for Computing Machinery, 2017, pp. 1825–1832. ISBN: 9781450349390. DOI: 10.1145/3067695.3084220.

[100] Jakub Husa. "Comparison of genetic programming methods on design of cryptographic boolean functions". In: *European Conference on Genetic Programming*. Springer. 2019, pp. 228–244. DOI: 10.1007/978-3-030-16670-0_15.

[101] Stjepan Picek, Claude Carlet, Sylvain Guilley, Julian F Miller, and Domagoj Jakobovic. "Evolutionary algorithms for boolean functions in diverse domains of cryptography". In: *Evolutionary computation* 24.4 (2016), pp. 667–694. DOI: 10.1162/EVCO_a_00190.

[102] Stjepan Picek, Elena Marchiori, Lejla Batina, and Domagoj Jakobovic. "Combining evolutionary computation and algebraic constructions to find cryptography-relevant Boolean functions". In: *Parallel Problem Solving from Nature–PPSN XIII: 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings 13*. Springer. 2014, pp. 822–831. DOI: 10.1007/978-3-319-10762-2_81.

[103] Luca Mariot, Domagoj Jakobovic, Alberto Leporati, and Stjepan Picek. "Hyper-bent Boolean functions and evolutionary algorithms". In: *European Conference on Genetic Programming*. Springer. 2019, pp. 262–277. DOI: 10.1007/978-3-030-16670-0_17.

[104] Luca Mariot, Stjepan Picek, Domagoj Jakobovic, Marko Djurasevic, and Alberto Leporati. "Evolutionary construction of perfectly balanced boolean functions". In: *2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2022, pp. 1–8. DOI: 10.1109/CEC55065.2022.9870427.

[105] Claude Carlet, Marko Djurasevic, Domagoj Jakobovic, Luca Mariot, and Stjepan Picek. "Evolving constructions for balanced, highly nonlinear boolean functions". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 1147–1155. ISBN: 9781450392372. DOI: 10.1145/3512290.3528871.

[106] Claude Carlet, Domagoj Jakobovic, and Stjepan Picek. "Evolutionary algorithms-assisted construction of cryptographic boolean functions". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '21. Lille, France: Association for Computing Machinery, 2021, pp. 565–573. ISBN: 9781450383509. DOI: 10.1145/3449639.3459362.

[107] L. F. Menabrea. "Sketch of the Analytical Engine invented by Charles Babbage, Esq." In: *Ada's Legacy: Cultures of Computing from the Victorian to the Digital Age*. Association for Computing Machinery and Morgan & Claypool, 2015. ISBN: 9781970001495. DOI: 10.1145/2809523.2809528.

[108] Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, et al. "Program synthesis". In: *Foundations and Trends® in Programming Languages* 4.1-2 (2017), pp. 1–119. DOI: 10.1561/2500000010.

[109] Zohar Manna and Richard J Waldinger. "Toward automatic program synthesis". In: *Communications of the ACM* 14.3 (1971), pp. 151–165. DOI: 10.1145/362566.362568.

[110] Zohar Manna and Richard Waldinger. "Knowledge and reasoning in program synthesis". In: *Artificial intelligence* 6.2 (1975), pp. 175–208. DOI: 10.1016/0004-3702(75)90008-9.

[111] Wolfgang Bibel. "Syntax-directed, semantics-supported program synthesis". In: *Artificial Intelligence* 14.3 (1980), pp. 243–261. DOI: 10.1016/0004-3702(80)90050-8.

[112] Martin Ward. "Proving program refinements and transformations". PhD thesis. University of Oxford PhD Thesis, 1989.

[113] Hans Hüttel. "On Program Synthesis and Large Language Models". In: *Commun. ACM* (Dec. 2024). Online First. ISSN: 0001-0782. DOI: 10.1145/3680410.

[114] Dominique Méry and Neeraj Kumar Singh. "Automatic code generation from Event-B models". In: *Proceedings of the 2nd Symposium on Information and Communication Technology*. 2011, pp. 179–188. DOI: 10.1145/2069216.2069252.

[115] Frank J. Budinsky, Marilyn A. Finnie, John M. Vlissides, and Patsy S. Yu. "Automatic code generation from design patterns". In: *IBM systems Journal* 35.2 (1996), pp. 151–171. DOI: 10.1147/sj.352.0151.

[116] A-E Rugina, David Thomas, Xavier Olive, and Guillaume Veran. "Gene-auto: Automatic software code generation for real-time embedded systems". In: *DASIA 2008-Data Systems In Aerospace* 665 (2008), p. 28. DOI: 2008ESASP.665E..28R.

[117] Hanwen Sun, Yuanping Nie, Xiang Li, Minhuan Huang, Jianwen Tian, and Wei Kong. "An Automatic Code Generation Method Based on Sequence Generative Adversarial Network". In: *2022 7th IEEE International Conference on Data Science in Cyberspace (DSC)*. IEEE. 2022, pp. 383–390. DOI: 10.1109/DSC55868.2022.00059.

[118] Tomas G Moreira, Marco A Wehrmeister, Carlos E Pereira, Jean-Francois Petin, and Eric Levrat. "Automatic code generation for embedded systems: From UML specifications to VHDL code". In: *2010 8th IEEE International Conference on Industrial Informatics*. IEEE. 2010, pp. 1085–1090. DOI: 10.1109/INDIN.2010.5549590.

[119] Zhiqiang Liu, Yong Dou, Jingfei Jiang, and Jinwei Xu. "Automatic code generation of convolutional neural networks in FPGA implementation". In: *2016 International conference on field-programmable technology (FPT)*. IEEE. 2016, pp. 61–68. DOI: 10.1109/FPT.2016.7929190.

[120] Gaetanino Paolone, Martina Marinelli, Romolo Paesani, and Paolino Di Felice. "Automatic code generation of MVC web applications". In: *Computers* 9.3 (2020), p. 56. DOI: 10.3390/computers9030056.

[121] Conor Ryan, John James Collins, and Michael O Neill. "Grammatical evolution: Evolving programs for an arbitrary language". In: *Genetic Programming: First European Workshop, EuroGP'98 Paris, France, April 14–15, 1998 Proceedings 1*. Springer. 1998, pp. 83–96. DOI: 10.1007/BFb0055930.

[122] Yasong Zhang, Yue Li, and Xiaoling Wang. "An optimized hybrid evolutionary algorithm for accelerating automatic code optimization". In: *Third International Seminar on Artificial Intelligence, Networking, and Information Technology (AINIT 2022)*. Vol. 12587. SPIE. 2023, pp. 488–496. DOI: 10.1117/12.2667392.

[123] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, et al. "Ansor: Generating High-Performance tensor programs for deep learning". In: *14th USENIX symposium on operating systems design and implementation (OSDI 20)*. 2020, pp. 863–879.

[124] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. "TVM: An automated End-to-End optimizing compiler for deep learning". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 578–594.

[125] Frode Eika Sandnes and Graham M Megson. "A hybrid genetic algorithm applied to automatic parallel controller code generation". In: *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*. IEEE. 1996, pp. 70–75. DOI: 10.1109/EMWRTS.1996.557799.

[126] James Alfred Walker, Yang Liu, Gianluca Tempesti, and Andy M Tyrrell. "Automatic code generation on a MOVE processor using Cartesian genetic programming". In: *Evolvable Systems: From Biology to Hardware: 9th International Conference, ICES 2010, York, UK, September 6-8, 2010. Proceedings 9*. Springer. 2010, pp. 238–249. DOI: 10.1007/978-3-642-15323-5_21.

[127] Wildor Ferrel Serruto and Luis Alfaro Casas. "Automatic Code Generation for Microcontroller-Based System Using Multi-objective Linear Genetic Programming". In: *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2017, pp. 279–285. DOI: 10.1109/CSCI.2017.47.

[128] Marlon Löppenberg and Andreas Schwung. "Self Optimisation and Automatic Code Generation by Evolutionary Algorithms in PLC based Controlling Processes". In: *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*. 2023, pp. 1–6. DOI: 10.1109/INDIN51400.2023.10218168.

[129] Ulya R Karpuzcu. "Automatic verilog code generation through grammatical evolution". In: *Proceedings of the 7th annual workshop on Genetic and evolutionary computation*. 2005, pp. 394–397. DOI: 10.1145/1102256.1102346.

[130] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[131] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805 [cs.CL].

[132] Anthony Gillioz, Jacky Casas, Elena Mugellini, and Omar Abou Khaled. "Overview of the Transformer-based Models for NLP Tasks". In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. IEEE. 2020, pp. 179–183. DOI: 10.15439/2020F20.

[133] Meta AI. *LLaMA: Open and Efficient Foundation Language Models*. 2023. DOI: 10.48550/arXiv.2302.13971. arXiv: 2302.13971 [cs.CL].

[134] Meta AI. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. DOI: 10.48550/arXiv.2307.09288. arXiv: 2307.09288 [cs.CL].

[135] Meta AI. *The Llama 3 Herd of Models*. 2024. DOI: 10.48550/arXiv.2407.21783.

[136] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[137] OpenAI. *GPT-4 Technical Report*. 2024. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs.CL].

[138] Aram Bahrini, Mohammadsadra Khamoshifar, Hossein Abbasimehr, Robert J. Riggs, Maryam Esmaeili, Rastin Mastali Majdabadkohne, and Morteza Pasehvar. "ChatGPT: Applications, Opportunities, and Threats". In: *2023 Systems and Information Engineering Design Symposium (SIEDS)*. 2023, pp. 274–279. DOI: 10.1109/SIEDS58326.2023.10137850.

[139] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. "Expectation vs Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models". In: *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*. CHI EA '22. New Orleans, LA, USA: Association for Computing Machinery, 2022. ISBN: 9781450391566. DOI: 10.1145/3491101.3519665.

[140] Zhijie Liu, Yutian Tang, Xiapu Luo, Yuming Zhou, and Liang Feng Zhang. "No Need to Lift a Finger Anymore? Assessing the Quality of Code Generation by ChatGPT". In: *IEEE Transactions on Software Engineering* 50.6 (2024), pp. 1548–1584. DOI: 10.1109/TSE.2024.3392499.

[141] Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. *LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation*. 2023. DOI: 10.48550/arXiv.2308.02828. arXiv: 2308.02828 [cs.SE].

[142] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. *Program Synthesis with Large Language Models*. 2021. DOI: 10.48550/arXiv.2108.07732. arXiv: 2108.07732 [cs.PL].

[143] Thomas Helmuth and Peter Kelly. "PSB2: the second program synthesis benchmark suite". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '21. Lille, France: Association for Computing Machinery, 2021, pp. 785–794. ISBN: 9781450383509. DOI: 10.1145/3449639.3459285.

[144] Thomas Helmuth and Peter Kelly. "Applying genetic programming to PSB2: the next generation program synthesis benchmark suite". In: *Genetic Programming and Evolvable Machines* 23.3 (2022), pp. 375–404. DOI: 10.1007/s10710-022-09434-y.

[145] Dominik Sobania, Martin Briesch, and Franz Rothlauf. "Choose your programming copilot: A comparison of the program synthesis performance of github copilot and genetic programming". In: *Proceedings of the genetic and evolutionary computation conference*. 2022, pp. 1019–1027. DOI: 10.1145/3512290.3528700.

[146] Dominik Sobania, Justyna Petke, Martin Briesch, and Franz Rothlauf. "A Comparison of Large Language Models and Genetic Programming for Program Synthesis". In: *IEEE Transactions on Evolutionary Computation* (2024), pp. 1–1. DOI: 10.1109/TEVC.2024.3410873.

[147] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. *Evaluating Large Language Models Trained on Code*. 2021. DOI: 10.48550/arXiv.2107.03374. arXiv: 2107.03374 [cs.LG].

[148] Leonardo Lucio Custode, Chiara Camilla Migliore Rambaldi, Marco Roveri, and Giovanni Iacca. "Comparing Large Language Models and Grammatical Evolution for Code Generation". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '24 Companion. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1830–1837. ISBN: 9798400704956. DOI: 10.1145/3638530.3664162.

[149] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation". In: *Advances in Neural Information Processing Systems* 36 (2024).

[150] Qinyun Wu, Chao Peng, Pengfei Gao, Ruida Hu, Haoyu Gan, Bo Jiang, Jinhe Tang, Zhiwen Deng, Zhanming Guan, Cuiyun Gao, Xia Liu, and Ping Yang. *RepoMasterEval: Evaluating Code Completion via Real-World Repositories*. 2024. DOI: 10.48550/arXiv.2408.03519. arXiv: 2408.03519 [cs.SE].

[151] Jia Li, Ge Li, Xuanming Zhang, Yihong Dong, and Zhi Jin. *EvoCodeBench: An Evolving Code Generation Benchmark Aligned with Real-World Code Repositories*. 2024. DOI: 10.48550/arXiv.2404.00599. arXiv: 2404.00599 [cs.CL].

[152] Ulyana Piterbarg, Lerrel Pinto, and Rob Fergus. *Training Language Models on Synthetic Edit Sequences Improves Code Synthesis*. 2024. DOI: 10.48550/arXiv.2410.02749. arXiv: 2410.02749 [cs.LG].

[153] William B Langdon. "Genetic improvement of programs". In: *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE. 2014, pp. 14–19. DOI: 10.1109/SYNASC.2014.10.

[154] William B. Langdon and Mark Harman. "Optimizing Existing Software With Genetic Programming". In: *IEEE Transactions on Evolutionary Computation* 19.1 (2015), pp. 118–135. DOI: 10.1109/TEVC.2013.2281544.

[155] William B. Langdon and Gabriela Ochoa. "Genetic improvement: A key challenge for evolutionary computation". In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. 2016, pp. 3068–3075. DOI: 10.1109/CEC.2016.7744177.

[156] William B. Langdon, Gabin An, Aymeric Blot, Vesna Nowack, Justyna Petke, Shin Yoo, Oliver Krauss, Erik M. Fredericks, and Daniel Blackwell. "The 13th International Workshop on Genetic Improvement(GI @ ICSE 2024)". In: *SIGSOFT Softw. Eng. Notes* 49.3 (July 2024), pp. 42–50. ISSN: 0163-5948. DOI: 10.1145/3672089.3672102.

[157] Justyna Petke, Mark Harman, William B Langdon, and Westley Weimer. "Using genetic improvement and code transplants to specialise a C++ program to a problem class". In: *Genetic Programming: 17th European Conference, EuroGP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers 17*. Springer. 2014, pp. 137–149. DOI: 10.1007/978-3-662-44303-3_12.

[158] Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. "Specialising Software for Different Downstream Applications Using Genetic Improvement and Code Transplantation". In: *IEEE Transactions on Software Engineering* 44.6 (2018), pp. 574–594. DOI: 10.1109/TSE.2017.2702606.

[159] Francesco Marino, Giovanni Squillero, and Alberto Tonda. "A general-purpose framework for genetic improvement". In: *Parallel Problem Solving from Nature–PPSN XIV: 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings 14*. Springer. 2016, pp. 345–352. DOI: 10.1007/978-3-319-45823-6_32.

[160] Michael Fenton, James McDermott, David Fagan, Stefan Forstenlechner, Erik Hemberg, and Michael O'Neill. "PonyGE2: grammatical evolution in Python". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '17. Berlin, Germany: Association for Computing Machinery, 2017, pp. 1194–1201. ISBN: 9781450349390. DOI: 10.1145/3067695.3082469.

[161] Gabin An, Aymeric Blot, Justyna Petke, and Shin Yoo. "PyGGI 2.0: language independent genetic improvement framework". In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Tallinn, Estonia: Association for Computing Machinery, 2019, pp. 1100–1104. ISBN: 9781450355728. DOI: 10.1145/3338906.3341184.

[162] Aymeric Blot and Justyna Petke. *MAGPIE: Machine Automated General Performance Improvement via Evolution of Software*. 2022. DOI: 10.48550/arXiv.2208.02811. arXiv: 2208.02811 [cs.SE].

[163] Nadia Alshahwan. "Industrial Experience of Genetic Improvement in Facebook". In: *2019 IEEE/ACM International Workshop on Genetic Improvement (GI)*. 2019, pp. 1–1. DOI: 10.1109/GI.2019.00010.

[164] James Callan, William Langdon, and Justyna Petke. "On Reducing Network Usage with Genetic Improvement". In: *Proceedings of the 13th ACM/IEEE International Workshop on Genetic Improvement*. GI '24. Lisbon, Portugal: Association for Computing Machinery, 2024, pp. 23–30. ISBN: 9798400705731. DOI: 10.1145/3643692.3648262.

[165] William Langdon and David Clark. "Deep Mutations have Little Impact". In: *Proceedings of the 13th ACM/IEEE International Workshop on Genetic Improvement*. GI '24. Lisbon, Portugal: Association for Computing Machinery, 2024, pp. 1–8. ISBN: 9798400705731. DOI: 10.1145/3643692.3648259.

[166] Vadim Liventsev, Anastasiia Grishina, Aki Härmä, and Leon Moonen. "Fully Autonomous Programming with Large Language Models". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '23. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 1146–1155. ISBN: 9798400701191. DOI: 10.1145/3583131.3590481.

[167] Michal Pluhacek, Anezka Kazikova, Tomas Kadavy, Adam Viktorin, and Roman Senkerik. "Leveraging Large Language Models for the Generation of Novel Metaheuristic Optimization Algorithms". In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. GECCO '23 Companion. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 1812–1820. ISBN: 9798400701207. DOI: 10.1145/3583133.3596401.

[168] Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. *Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap*. 2024. DOI: 10.48550/arXiv.2401.10034. arXiv: 2401.10034 [cs.NE].

[169] Giovanni Pinna, Damiano Ravalico, Luigi Rovito, Luca Manzoni, and Andrea De Lorenzo. "Enhancing Large Language Models-Based Code Generation by Leveraging Genetic Improvement". In: *Genetic Programming*. Ed. by Mario Giacobini, Bing Xue, and Luca Manzoni. Cham: Springer Nature Switzerland, 2024, pp. 108–124. ISBN: 978-3-031-56957-9. DOI: 10.1007/978-3-031-56957-9_7.

[170] Jean-Baptiste Mouret. *Large language models help computer programs to evolve*. 2024. DOI: 10.1038/d41586-023-03998-0.

[171] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. "Mathematical discoveries from program search with large language models". In: *Nature* 625.7995 (2024), pp. 468–475. DOI: 10.1038/s41586-023-06924-6.

[172] Erik Hemberg, Stephen Moskal, and Una-May O'Reilly. "Evolving code with a large language model". In: *Genetic Programming and Evolvable Machines* 25.2 (2024), p. 21. DOI: 10.1007/s10710-024-09494-2.

[173] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. "Evolution through large models". In: *Handbook of Evolutionary Machine Learning*. Springer, 2023, pp. 331–366. DOI: 10.1007/978-981-99-3814-8_11.

[174] Ning Tao, Anthony Ventresque, Vivek Nallur, and Takfarinas Saber. "Enhancing Program Synthesis with Large Language Models Using Many-Objective Grammar-Guided Genetic Programming". In: *Algorithms* 17.7 (2024), p. 287. DOI: 10.3390/a17070287.

[175] Fei Liu, Xi Lin, Zhenkun Wang, Shunyu Yao, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. *Large Language Model for Multi-objective Evolutionary Optimization*. 2024. DOI: 10.48550/arXiv.2310.12541. arXiv: 2310.12541 [cs.NE].

[176] Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. *Algorithm Evolution Using Large Language Model*. 2023. DOI: 2311.15249. arXiv: 2311.15249 [cs.NE].

[177] Niki van Stein and Thomas Bäck. *LLaMEA: A Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics*. 2024. DOI: 10.48550/arXiv.2405.20132. arXiv: 2405.20132 [cs.NE].

[178] Shuvayan Brahmachary, Subodh M. Joshi, Aniruddha Panda, Kaushik Koneripalli, Arun Kumar Sagotra, Harshil Patel, Ankush Sharma, Ameya D. Jagtap, and Kaushic Kalyanaraman. *Large Language Model-Based Evolutionary Optimizer: Reasoning with elitism*. 2024. DOI: 10.48550/arXiv.2403.02054. arXiv: 2403.02054 [cs.AI].

[179] Paula Maddigan, Andrew Lensen, and Bing Xue. *Explaining Genetic Programming Trees using Large Language Models*. 2024. DOI: 10.48550/arXiv.2403.03397. arXiv: 2403.03397 [cs.NE].

[180] Leonardo Lucio Custode, Fabio Caraffini, Anil Yaman, and Giovanni Iacca. "An investigation on the use of Large Language Models for hyperparameter tuning in Evolutionary Algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '24 Companion. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1838–1845. ISBN: 9798400704956. DOI: 10.1145/3638530.3664163.

[181] Steven Jorgensen, Giorgia Nadizar, Gloria Pietropolli, Luca Manzoni, Eric Medvet, Una-May O'Reilly, and Erik Hemberg. "Large Language Model-based Test Case Generation for GP Agents". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '24. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 914–923. ISBN: 9798400704949. DOI: 10.1145/3638529.3654056.

[182] Martina Saletta and Claudio Ferretti. "Exploring the Prompt Space of Large Language Models through Evolutionary Sampling". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '24. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 1345–1353. ISBN: 9798400704949. DOI: 10.1145/3638529.3654049.

[183] Angelica Chen, David Dohan, and David So. "EvoPrompting: language models for code-level neural architecture search". In: *Advances in Neural Information Processing Systems* 36 (2024).

[184] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. *Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers*. 2024. DOI: 10.48550/arXiv.2309.08532. arXiv: 2309.08532 [cs.CL].

[185] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. *Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution*. 2023. DOI: 10.48550/arXiv.2309.16797. arXiv: 2309.16797 [cs.CL].

[186] Amina Adadi and Mohammed Berrada. "Peeking inside the black-box: a survey on explainable artificial intelligence (XAI)". In: *IEEE access* 6 (2018), pp. 52138–52160. DOI: 10.1109/ACCESS.2018.2870052.

[187] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini, and Fosca Giannotti. "Local rule-based explanations of black box decision systems". In: *arXiv preprint arXiv:1805.10820* (2018). DOI: 10.48550/arXiv.1805.10820.

[188] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[189] Christoph Molnar, Gunnar König, Julia Herbinger, Timo Freiesleben, Susanne Dandl, Christian A Scholbeck, Giuseppe Casalicchio, Moritz Grosse-Wentrup, and Bernd Bischl. "Pitfalls to avoid when interpreting machine learning models". In: (2020).

[190] Michaela Benk and Andrea Ferrario. "Explaining Interpretable Machine Learning: Theory, Methods and Applications". In: *Methods and Applications (December 11, 2020)* (2020).

[191] Joshua James Hatherley. "Limits of trust in medical AI". In: *Journal of medical ethics* 46.7 (2020), pp. 478–481. DOI: 10.1136/medethics-2019-105935.

[192] Andreas Holzinger. "From Machine Learning to Explainable AI". In: *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*. 2018, pp. 55–66. DOI: 10.1109/DISA.2018.8490530.

[193] Derek Doran, Sarah Schulz, and Tarek R. Besold. "What Does Explainable AI Really Mean? A New Conceptualization of Perspectives". In: *CoRR* abs/1710.00794 (2017). arXiv: 1710.00794. URL: http://arxiv.org/abs/1710.00794.

[194] Randy Goebel, Ajay Chander, Katharina Holzinger, Freddy Lecue, Zeynep Akata, Simone Stumpf, Peter Kieseberg, and Andreas Holzinger. "Explainable AI: The New 42?" In: *Machine Learning and Knowledge Extraction*. Ed. by Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl. Cham: Springer International Publishing, 2018, pp. 295–303. ISBN: 978-3-319-99740-7. DOI: 10.1007/978-3-319-99740-7_21.

[195] Robert R Hoffman, Shane T Mueller, Gary Klein, and Jordan Litman. "Metrics for explainable AI: Challenges and prospects". In: *arXiv preprint arXiv:1812.04608* (2018). DOI: 10.48550/arXiv.1812.04608.

[196] Hani Hagras. "Toward human-understandable, explainable AI". In: *Computer* 51.9 (2018), pp. 28–36. DOI: 10.1109/MC.2018.3620965.

[197] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "" Why should i trust you?" Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144. DOI: 10.1145/2939672.2939778.

[198] Giulia Vilone and Luca Longo. "Explainable artificial intelligence: a systematic review". In: *arXiv preprint arXiv:2006.00093* (2020). DOI: 10.48550/arXiv.2006.00093.

[199] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. "Explainable AI: A brief survey on history, research areas, approaches and challenges". In: *CCF international conference on natural language processing and Chinese computing*. Springer. 2019, pp. 563–574. DOI: 10.1007/978-3-030-32236-6_51.

[200] Omer Sagi and Lior Rokach. "Explainable decision forest: Transforming a decision forest into an interpretable tree". In: *Information Fusion* 61 (2020), pp. 124–138. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2020.03.013.

[201] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. "Explainable artificial intelligence: A survey". In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2018, pp. 0210–0215. DOI: 10.23919/MIPRO.2018.8400040.

[202] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. "A survey of methods for explaining black box models". In: *ACM computing surveys (CSUR)* 51.5 (2018), pp. 1–42. DOI: 10.1145/3236009.

[203] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. "An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models". In: *Decision Support Systems* 51.1 (2011), pp. 141–154. DOI: 10.1016/j.dss.2010.12.003.

[204] Wojciech Samek and Klaus-Robert Müller. "Towards explainable artificial intelligence". In: *Explainable AI: interpreting, explaining and visualizing deep learning*. Springer, 2019, pp. 5–22. DOI: 10.1007/978-3-030-28954-6_1.

[205] Arun Das and Paul Rad. "Opportunities and challenges in explainable artificial intelligence (xai): A survey". In: *arXiv preprint arXiv:2006.11371* (2020). DOI: 10.48550/arXiv.2006.11371.

[206] Boris Kovalerchuk, Muhammad Aurangzeb Ahmad, and Ankur Teredesai. "Survey of explainable machine learning with visual and granular methods beyond quasi-explanations". In: *Interpretable artificial intelligence: A perspective of granular computing* (2021), pp. 217–267. DOI: 10.1007/978-3-030-64949-4_8.

[207] Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. "On explaining decision trees". In: *arXiv preprint arXiv:2010.11034* (2020). DOI: 10.48550/arXiv.2010.11034.

[208] Leonard A Breslow and David W Aha. "Simplifying decision trees: A survey". In: *The Knowledge Engineering Review* 12.01 (1997), pp. 1–40.

[209] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. "Interpretable Decision Sets: A Joint Framework for Description and Prediction". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1675–1684. ISBN: 9781450342322. DOI: 10.1145/2939672.2939874.

[210] Guolong Su, Dennis Wei, Kush R Varshney, and Dmitry M Malioutov. "Interpretable two-level boolean rule learning for classification". In: *arXiv preprint arXiv:1511.07361* (2015). DOI: 10.48550/arXiv.1511.07361.

[211] Hui Zou and Trevor Hastie. "Regularization and variable selection via the elastic net". In: *Journal of the royal statistical society: series B (statistical methodology)* 67.2 (2005), pp. 301–320. DOI: 10.1111/j.1467-9868.2005.00503.x.

[212] Berk Ustun and Cynthia Rudin. "Supersparse linear integer models for optimized medical scoring systems". In: *Machine Learning* 102.3 (2016), pp. 349–391. DOI: 10.1007/s10994-015-5528-6.

[213] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288. DOI: 10.1111/j.2517-6161.1996.tb02080.x.

[214] Forough Poursabzi-Sangdeh, Daniel G Goldstein, Jake M Hofman, Jennifer Wortman Wortman Vaughan, and Hanna Wallach. "Manipulating and Measuring Model Interpretability". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI '21. Yokohama, Japan: Association for Computing Machinery, 2021. ISBN: 9781450380966. DOI: 10.1145/3411764.3445315.

[215] Jaume Bacardit, Alexander E. I. Brownlee, Stefano Cagnoni, Giovanni Iacca, John McCall, and David Walker. "The intersection of evolutionary computation and explainable AI". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 1757–1762. ISBN: 9781450392686. DOI: 10.1145/3520304.3533974.

[216] Alberto Fernandez, Francisco Herrera, Oscar Cordon, Maria Jose del Jesus, and Francesco Marcelloni. "Evolutionary fuzzy systems for explainable artificial intelligence: Why, when, what for, and where to?" In: *IEEE Computational intelligence magazine* 14.1 (2019), pp. 69–81. DOI: 10.1109/MCI.2018.2881645.

[217] Shubham Sharma, Jette Henderson, and Joydeep Ghosh. "Certifai: Counterfactual explanations for robustness, transparency, interpretability, and fairness of artificial intelligence models". In: *arXiv preprint arXiv:1905.07857* (2019). DOI: 10.1145/3375627.3375812.

[218] Andrea Ferigo, Leonardo Lucio Custode, and Giovanni Iacca. "Quality Diversity Evolutionary Learning of Decision Trees". In: *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*. SAC '23. Tallinn, Estonia: Association for Computing Machinery, 2023, pp. 425–432. ISBN: 9781450395175. DOI: 10.1145/3555776.3577591.

[219] Ryan J Urbanowicz and Jason H Moore. "Learning classifier systems: a complete introduction, review, and roadmap". In: *Journal of Artificial Evolution and Applications* 2009 (2009). DOI: 10.1155/2009/736398.

[220] Luigi Rovito, Lorenzo Bonin, Luca Manzoni, and Andrea De Lorenzo. "An Evolutionary Computation Approach for Twitter Bot Detection". In: *Applied Sciences* 12.12 (2022), p. 5915. DOI: 10.3390/app12125915.

[221] Alberto Cano, Amelia Zafra, and Sebastián Ventura. "An interpretable classification rule mining algorithm". In: *Information Sciences* 240 (2013), pp. 1–20. DOI: 10.1016/j.ins.2013.03.038.

[222] Aniko Ekart and Sandor Z. Nemeth. "Selection based on the pareto nondomination criterion for controlling code growth in genetic programming". In: *Genetic Programming and Evolvable Machines* 2.1 (2001), pp. 61–73. DOI: 10.1023/A:1010070616149.

[223] Guido F Smits and Mark Kotanchek. "Pareto-front exploitation in symbolic regression". In: *Genetic programming theory and practice II*. Springer, 2005, pp. 283–299. DOI: 10.1007/0-387-23254-0_17.

[224] Andrew Lensen. "Mining Feature Relationships in Data". In: *European Conference on Genetic Programming (Part of EvoStar)*. Springer. 2021, pp. 247–262. DOI: 10.1007/978-3-030-72812-0_16.

[225] Aidan Murphy, Gráinne Murphy, Jorge Amaral, Douglas MotaDias, Enrique Naredo, and Conor Ryan. "Towards incorporating human knowledge in fuzzy pattern tree evolution". In: *European Conference on Genetic Programming (Part of EvoStar)*. Springer. 2021, pp. 66–81. DOI: 10.1007/978-3-030-72812-0_5.

[226] Marco Virgolin, Tanja Alderliesten, and Peter AN Bosman. "On explaining machine learning models by evolving crucial and compact features". In: *Swarm and Evolutionary Computation* 53 (2020), p. 100640. DOI: 10.1016/j.swevo.2019.100640.

[227] Karina Brotto Rebuli, Mario Giacobini, Sara Silva, and Leonardo Vanneschi. "A Comparison of Structural Complexity Metrics for Explainable Genetic Programming". In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. 2023, pp. 539–542. DOI: 10.1145/3583133.3590595.

[228] Daniel Hein, Steffen Udluft, and Thomas A Runkler. "Interpretable policies for reinforcement learning by genetic programming". In: *Engineering Applications of Artificial Intelligence* 76 (2018), pp. 158–169. DOI: 10.1016/j.engappai.2018.09.007.

[229] Eric Medvet, Alberto Bartoli, Barbara Carminati, and Elena Ferrari. "Evolutionary inference of attribute-based access control policies". In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2015, pp. 351–365. DOI: 10.1007/978-3-319-15934-8_24.

[230] Noman Javed, Fernand R. Gobet, and Peter Lane. "Simplification of genetic programs: a literature survey". In: *Data Min. Knowl. Discov.* 36 (2022), pp. 1279–1300. DOI: 10.1007/s10618-022-00830-7.

[231] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285. DOI: 10.1613/jair.301.

[232] Dennis G Wilson, Sylvain Cussat-Blanc, Hervé Luga, and Julian F Miller. "Evolving simple programs for playing atari games". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '18. Kyoto, Japan: Association for Computing Machinery, 2018, pp. 229–236. ISBN: 9781450356183. DOI: 10.1145/3205455.3205578.

[233] Mathurin Videau, Alessandro Leite, Olivier Teytaud, and Marc Schoenauer. "Multi-objective genetic programming for explainable reinforcement learning". In: *European Conference on Genetic Programming (Part of EvoStar)*. Springer. 2022, pp. 278–293. DOI: 10.1007/978-3-031-02056-8_18.

[234] Giorgia Nadizar, Luigi Rovito, Dennis G. Wilson, and Eric Medvet. "Interpretable Control Competition". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '24 Companion. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 11–12. ISBN: 9798400704956. DOI: 10.1145/3638530.3664051.

[235] Giorgia Nadizar, Eric Medvet, and Dennis Wilson. "Searching for a Diversity of Interpretable Graph Control Policies". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '24. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 933–941. ISBN: 9798400704949. DOI: 10.1145/3638529.3653987.

[236] Giorgia Nadizar, Eric Medvet, and Dennis G. Wilson. "Naturally Interpretable Control Policies via Graph-Based Genetic Programming". In: *Genetic Programming*. Ed. by Mario Giacobini, Bing Xue, and Luca Manzoni. Cham: Springer Nature Switzerland, 2024, pp. 73–89. ISBN: 978-3-031-56957-9. DOI: 10.1007/978-3-031-56957-9_5.

[237] Leonardo L. Custode and Giovanni Iacca. "Evolutionary Learning of Interpretable Decision Trees". In: *IEEE Access* 11 (2023), pp. 6169–6184. DOI: 10.1109/ACCESS.2023.3236260.

[238] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3 (1992), pp. 279–292. DOI: 10.1007/BF00992698.

[239] Leonardo Lucio Custode and Giovanni Iacca. "A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces". In: *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2021, pp. 1–8. DOI: 10.1109/SSCI50451.2021.9660048.

[240] Leonardo Lucio Custode and Giovanni Iacca. "Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 224–227. ISBN: 9781450392686. DOI: 10.1145/3520304.3528897.

[241] Varun Raj Kompella, Roberto Capobianco, Stacy Jong, Jonathan Browne, Spencer J. Fox, Lauren Ancel Meyers, Peter R. Wurman, and Peter Stone. "Reinforcement Learning for Optimization of COVID-19 Mitigation policies". In: *CoRR* abs/2010.10560 (2020). DOI: 2010.10560.

[242] Alexander Trott, Sunil Srinivasa, Douwe van der Wal, Sebastien Haneuse, and Stephan Zheng. "Building a Foundation for Data-Driven, Interpretable, and Robust Policy Design using the AI Economist". In: *CoRR* abs/2108.02904 (2021). DOI: 2108.02904.

[243] Risto Miikkulainen, Olivier Francon, Elliot Meyerson, Xin Qiu, Darren Sargent, Elisa Canzani, and Babak Hodjat. "From prediction to prescription: evolutionary optimization of nonpharmaceutical interventions in the COVID-19 pandemic". In: *IEEE Transactions on Evolutionary Computation* 25.2 (2021), pp. 386–401. DOI: 10.1109/TEVC.2021.3063217.

[244] Leonardo Lucio Custode and Giovanni Iacca. "Interpretable AI for policy-making in pandemics". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 1763–1769. ISBN: 9781450392686. DOI: 10.1145/3520304.3533959.

[245] Yazhou Yang and Marco Loog. "Active learning using uncertainty information". In: *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE. 2016, pp. 2646–2651. DOI: 10.1109/ICPR.2016.7900034.

[246] Burr Settles. "Active learning literature survey". In: (2009). URL: http://digital.library.wisc.edu/1793/60660.

[247] Zahra Mahoor, Jack Felag, and Josh Bongard. "Morphology dictates a robot's ability to ground crowd-proposed language". In: *arXiv preprint arXiv:1712.05881* (2017). DOI: 10.48550/arXiv.1712.05881.

[248] Jimmy Secretan, Nicholas Beato, David B D'Ambrosio, Adelein Rodriguez, Adam Campbell, Jeremiah T Folsom-Kovarik, and Kenneth O Stanley. "Picbreeder: A case study in collaborative evolutionary exploration of design space". In: *Evolutionary computation* 19.3 (2011), pp. 373–403. DOI: 10.1162/EVCO_a_00030.

[249] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. "Deep reinforcement learning from human preferences". In: *Advances in neural information processing systems* 30 (2017).

[250] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. "Active learning of regular expressions for entity extraction". In: *IEEE Transactions on Cybernetics* 48.3 (2017), pp. 1067–1080. DOI: 10.1109/TCYB.2017.2680466.

[251] Junio De Freitas, Gisele L Pappa, Altigran S da Silva, Marcos A Gonc, Edleno Moura, Adriano Veloso, Alberto HF Laender, Moisés G de Carvalho, et al. "Active learning genetic programming for record deduplication". In: *IEEE Congress on Evolutionary Computation*. IEEE. 2010, pp. 1–8. DOI: 10.1109/CEC.2010.5586104.

[252] Robert Isele and Christian Bizer. "Active learning of expressive linkage rules using genetic programming". In: *Journal of web semantics* 23 (2013), pp. 2–15. DOI: 10.1016/j.websem.2013.06.001.

[253] J von Neumann. "Theory of self-reproducing automata". In: *Mathematics of Computation* 21 (1966), p. 745.

[254] Edgar F Codd. *Cellular automata*. Academic press, 1968.

[255] Palash Sarkar. "A brief history of cellular automata". In: *Acm computing surveys (csur)* 32.1 (2000), pp. 80–107. DOI: 10.1145/349194.349202.

[256] Enrique Alba and Bernabé Dorronsoro. "The exploration/exploitation tradeoff in dynamic cellular genetic algorithms". In: *IEEE transactions on evolutionary computation* 9.2 (2005), pp. 126–142. DOI: 10.1109/TEVC.2005.843751.

[257] Enrique Alba and Bernabè Dorronsoro. "Introduction to Cellular Genetic Algorithms". In: *Cellular Genetic Algorithms*. Boston, MA: Springer US, 2008, pp. 3–20. ISBN: 978-0-387-77610-1. DOI: 10.1007/978-0-387-77610-1_1.

[258] Carolina Salto and Enrique Alba. "Cellular genetic algorithms: Understanding the behavior of using neighborhoods". In: *Applied Artificial Intelligence* 33.10 (2019), pp. 863–880. DOI: 10.1080/08839514.2019.1646005.

[259] Yanlan Deng, Juxia Xiong, and Qiuhong Wang. "A hybrid cellular genetic algorithm for the traveling salesman problem". In: *Mathematical Problems in Engineering* 2021 (2021), pp. 1–16. DOI: 10.1155/2021/6697598.

[260] Tadahiko Murata and Mitsuo Gen. "Cellular genetic algorithm for multi-objective optimization". In: *Proc. of the 4th Asian Fuzzy System Symposium*. Citeseer. 2002, pp. 538–542.

[261] Antonio J Nebro, Juan J Durillo, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. "Mocell: A cellular genetic algorithm for multiobjective optimization". In: *International Journal of Intelligent Systems* 24.7 (2009), pp. 726–746. DOI: 10.1002/int.20358.

[262] Luca Mariot, Stjepan Picek, Domagoj Jakobovic, and Alberto Leporati. "Evolutionary algorithms for the design of orthogonal latin squares based on cellular automata". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '17. Berlin, Germany: Association for Computing Machinery, 2017, pp. 306–313. ISBN: 9781450349208. DOI: 10.1145/3071178.3071284.

[263] Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano. "A scalable cellular implementation of parallel genetic programming". In: *IEEE Transactions on Evolutionary Computation* 7.1 (2003), pp. 37–53. DOI: 10.1109/TEVC.2002.806168.

[264] Worthy N Martin. "Island (migration) models: evolutionary algorithms based on punctuated equilibria". In: *Handbook of evolutionary computation* (1997).

[265] Peter A Whigham and Grant Dick. "Implicitly controlling bloat in genetic programming". In: *IEEE Transactions on Evolutionary Computation* 14.2 (2009), pp. 173–190. DOI: 10.1109/TEVC.2009.2027314.

[266] Grant Dick and Peter A Whigham. "Controlling bloat through parsimonious elitist replacement and spatial structure". In: *European Conference on Genetic Programming*. Springer. 2013, pp. 13–24. DOI: 10.1007/978-3-642-37207-0_2.

[267] Yang Shi, Hongcheng Liu, Liang Gao, and Guohui Zhang. "Cellular particle swarm optimization". In: *Information Sciences* 181.20 (2011), pp. 4460–4493. DOI: 10.1016/j.ins.2010.05.025.

[268] Mohammed Azmi Al-Betar, Ahamad Tajudin Khader, Mohammed A Awadallah, Mahmmoud Hafsaldin Alawan, and Belal Zaqaibeh. "Cellular harmony search for optimization problems". In: *Journal of Applied Mathematics* 2013 (2013). DOI: 10.1155/2013/139464.

[269] Ming Zhang, Na Tian, Vasile Palade, Zhicheng Ji, and Yan Wang. "Cellular artificial bee colony algorithm with Gaussian distribution". In: *Information Sciences* 462 (2018), pp. 374–401. DOI: 10.1016/j.ins.2018.06.032.

[270] Antonio Della Cioppa, Angelo Marcelli, and Prisco Napoli. "Speciation in evolutionary algorithms: adaptive species discovery". In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. Dublin, Ireland: Association for Computing Machinery, 2011, pp. 1053–1060. ISBN: 9781450305570. DOI: 10.1145/2001576.2001719.

[271] Kenneth O Stanley and Risto Miikkulainen. "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2 (2002), pp. 99–127. DOI: 10.1162/106365602320169811.

[272] Leonardo Trujillo, Luis Muñoz, Edgar Galván-López, and Sara Silva. "neat genetic programming: Controlling bloat naturally". In: *Information Sciences* 333 (2016), pp. 21–43. DOI: 10.1016/j.ins.2015.11.010.

[273] Perla Juárez-Smith, Leonardo Trujillo, Mario García-Valdez, Francisco Fernández de Vega, and Francisco Chávez. "Local search in speciation-based bloat control for genetic programming". In: *Genetic Programming and Evolvable Machines* 20 (2019), pp. 351–384. DOI: 10.1007/s10710-019-09351-7.

[274] Sylvain Cussat-Blanc, Kyle Harrington, and Jordan Pollack. "Gene regulatory network evolution through augmenting topologies". In: *IEEE Transactions on Evolutionary Computation* 19.6 (2015), pp. 823–837. DOI: 10.1109/TEVC.2015.2396199.

[275] Tiago Mousinho Martins and Rui Ferreira Neves. "Applying genetic algorithms with speciation for optimization of grid template pattern detection in financial markets". In: *Expert Systems with Applications* 147 (2020), p. 113191. DOI: 10.1016/j.eswa.2020.113191.

[276] Ryan Wickman, Bibek Poudel, Taylor Michael Villarreal, Xiaofei Zhang, and Weizi Li. "Efficient Quality-Diversity Optimization through Diverse Quality Species". In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. GECCO '23 Companion. Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 699–702. ISBN: 9798400701207. DOI: 10.1145/3583133.3590581.

[277] Gloria Pietropolli, Stefano Nichele, and Eric Medvet. "The Role of the Substrate in CA-based Evolutionary Algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '24. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 768–777. ISBN: 9798400704949. DOI: 10.1145/3638529.3654112.

[278] David E Goldberg and Kalyanmoy Deb. "A comparative analysis of selection schemes used in genetic algorithms". In: *Foundations of genetic algorithms*. Vol. 1. Elsevier, 1991, pp. 69–93. DOI: 10.1016/B978-0-08-050684-5.50008-2.

[279] Huayang Xie and Mengjie Zhang. "Impacts of sampling strategies in tournament selection for genetic programming". In: *Soft computing* 16 (2012), pp. 615–633. DOI: 10.1007/s00500-011-0760-x.

[280] Jayshree Sarma and Kenneth De Jong. "An analysis of the effects of neighborhood size and shape on local selection algorithms". In: *International Conference on Parallel Problem Solving From Nature*. Springer. 1996, pp. 236–244. DOI: 10.1007/3-540-61723-X_988.

[281] Mario Giacobini, Marco Tomassini, Andrea GB Tettamanzi, and Enrique Alba. "Selection intensity in cellular evolutionary algorithms for regular lattices". In: *IEEE Transactions on Evolutionary Computation* 9.5 (2005), pp. 489–505. DOI: 10.1109/TEVC.2005.850298.

[282] Luigi Rovito, Andrea De Lorenzo, and Luca Manzoni. "Evolution of Walsh Transforms with Genetic Programming". In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. ACM, 2023, pp. 2386–2389. DOI: 10.1145/3583133.3596317.

[283] Luigi Rovito, Andrea De Lorenzo, and Luca Manzoni. "Discovering Non-Linear Boolean Functions by Evolving Walsh Transforms with Genetic Programming". In: *Algorithms* 16.11 (2023), p. 499. DOI: 10.3390/a16110499.

[284] Gilbert S Vernam. "Cipher printing telegraph systems: For secret wire and radio telegraphic communications". In: *Journal of the AIEE* 45.2 (1926), pp. 109–115. DOI: 10.1109/JAIEE.1926.6534724.

[285] Elaine B Barker, John Michael Kelsey, et al. *Recommendation for random number generation using deterministic random bit generators (revised)*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, 2007.

[286] Richard Peirce Brent. "Some long-period random number generators using shifts and xors". In: *ANZIAM Journal* 48 (2006), pp. C188–C202. DOI: 10.21914/anziamj.v48i0.40.

[287] Thomas W. Cusick and Pantelimon Stanica. "Chapter 2 - Fourier Analysis of Boolean Functions". In: *Cryptographic Boolean Functions and Applications (Second Edition)*. Ed. by Thomas W. Cusick and Pantelimon Stanica. Second Edition. Academic Press, 2017, pp. 7–29. ISBN: 978-0-12-811129-1. DOI: 10.1016/B978-0-12-811129-1.00002-X.

[288] Claude E Shannon. "Communication theory of secrecy systems". In: *The Bell system technical journal* 28.4 (1949), pp. 656–715. DOI: 10.1002/j.1538-7305.1949.tb00928.x.

[289] Matthew Becker and Ahmed Desoky. "A study of the DVD content scrambling system (CSS) algorithm". In: *Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology, 2004.* IEEE. 2004, pp. 353–356. DOI: 10.1109/ISSPIT.2004.1433792.

[290] Gérard D Cohen and Simon N Litsyn. "On the covering radius of Reed-Muller codes". In: *Discrete Mathematics* 106 (1992), pp. 147–155. DOI: 10.1016/0012-365X(92)90542-N.

[291] Xiang-dong Hou. "On the norm and covering radius of the first-order Reed-Muller codes". In: *IEEE Transactions on Information Theory* 43.3 (1997), pp. 1025–1027. DOI: 10.1109/18.568715.

[292] J. Blank and K. Deb. "pymoo: Multi-Objective Optimization in Python". In: *IEEE Access* 8 (2020), pp. 89497–89509. DOI: 10.1109/ACCESS.2020.2990567.

[293] Frank Wilcoxon. *Individual comparisons by ranking methods*. Springer, 1992. DOI: 10.1007/978-1-4612-4380-9_16.

[294] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. "Alpaca: A strong, replicable instruction-following model". In: *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html* 3.6 (2023), p. 7.

[295] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. *Self-Instruct: Aligning Language Models with Self-Generated Instructions*. 2023. DOI: 10.48550/arXiv.2212.10560. arXiv: 2212.10560 [cs.CL].

[296] Meta AI. *Code Llama: Open Foundation Models for Code*. 2024. DOI: 10.48550/arXiv.2308.12950. arXiv: 2308.12950 [cs.CL].

[297] Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. *LLM Critics Help Catch LLM Bugs*. 2024. DOI: 10.48550/arXiv.2407.00215. arXiv: 2407.00215 [cs.SE].

[298] Maarten Grootendorst. "KeyBERT: Minimal keyword extraction with BERT". In: *Zenodo* (2020).

[299] Thomas Helmuth and Lee Spector. "Problem-solving benefits of down-sampled lexicase selection". In: *Artificial life* 27.3–4 (2022), pp. 183–203. DOI: 10.1162/artl_a_00341.

[300] Thomas Helmuth and Lee Spector. "Explaining and exploiting the advantages of down-sampled lexicase selection". In: *Artificial Life Conference Proceedings 32*. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ... 2020, pp. 341–349. DOI: 10.1162/isal_a_00334.

[301] Ryan Boldi, Ashley Bao, Martin Briesch, Thomas Helmuth, Dominik Sobania, Lee Spector, and Alexander Lalejini. "A Comprehensive Analysis of Down-sampling for Genetic Programming-based Program Synthesis". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '24 Companion. Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 487–490. ISBN: 9798400704956. DOI: 10.1145/3638530.3654134.

[302] Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector. "Lexicase Selection for Program Synthesis: A Diversity Analysis". In: *Genetic Programming Theory and Practice XIII*. Ed. by Rick Riolo, W.P. Worzel, Mark Kotanchek, and Arthur Kordon. Cham: Springer International Publishing, 2016, pp. 151–167. ISBN: 978-3-319-34223-8. DOI: 10.1007/978-3-319-34223-8_9.

[303] Blossom Metevier, Anil Kumar Saini, and Lee Spector. "Lexicase Selection Beyond Genetic Programming". In: *Genetic Programming Theory and Practice XVI*. Ed. by Wolfgang Banzhaf, Lee Spector, and Leigh Sheneman. Cham: Springer International Publishing, 2019, pp. 123–136. ISBN: 978-3-030-04735-1. DOI: 10.1007/978-3-030-04735-1_7.

[304] Thomas Helmuth and William La Cava. "Lexicase selection". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 1385–1397. ISBN: 9781450392686. DOI: 10.1145/3520304.3533633.

[305] Li Ding, Ryan Boldi, Thomas Helmuth, and Lee Spector. "Lexicase selection at scale". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 2054–2062. ISBN: 9781450392686. DOI: 10.1145/3520304.3534026.

[306] H. B. Mann and D. R. Whitney. "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other". In: *The Annals of Mathematical Statistics* 18.1 (1947), pp. 50–60. ISSN: 00034851. URL: https://www.jstor.org/stable/2236101 (visited on 01/17/2024).

[307] William H Kruskal and W Allen Wallis. "Use of ranks in one-criterion variance analysis". In: *Journal of the American statistical Association* 47.260 (1952), pp. 583–621. DOI: 10.1080/01621459.1952.10483441.

[308] Sture Holm. "A simple sequentially rejective multiple test procedure". In: *Scandinavian journal of statistics* (1979), pp. 65–70. URL: https://www.jstor.org/stable/4615733.

[309] Ekaterina J. Vladislavleva, Guido F. Smits, and Dick den Hertog. "Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming". In: *IEEE Transactions on Evolutionary Computation* 13.2 (2009), pp. 333–349. DOI: 10.1109/TEVC.2008.926486.

[310] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[311] Michael O'Neill. *Riccardo Poli, William B. Langdon, Nicholas F. McPhee: a field guide to genetic programming*. 2009. DOI: 10.1007/s10710-008-9073-y.

[312] Sean Luke and Liviu Panait. "A survey and comparison of tree generation algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. Citeseer. 2001, pp. 81–88. DOI: 10.5555/2955239.2955250.

[313] William B Langdon, Riccardo Poli, Nicholas F McPhee, and John R Koza. "Genetic programming: An introduction and tutorial, with a survey of techniques and applications". In: *Computational intelligence: A compendium* (2008), pp. 927–1028. DOI: 10.1007/978-3-540-78293-3_22.

[314] Dazhuang Liu, Marco Virgolin, Tanja Alderliesten, and Peter A. N. Bosman. "Evolvability degeneration in multi-objective genetic programming for symbolic regression". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 973–981. ISBN: 9781450392372. DOI: 10.1145/3512290.3528787.

[315] Dazhuang Liu, Marco Virgolin, Tanja Alderliesten, and Peter AN Bosman. "Evolvability Degeneration in Multi-Objective Genetic Programming for Symbolic Regression". In: *arXiv preprint arXiv:2202.06983* (2022).

[316] Maarten Keijzer. "Improving symbolic regression with interval arithmetic and linear scaling". In: *European Conference on Genetic Programming*. Springer. 2003, pp. 70–82. DOI: 10.1007/3-540-36599-0_7.

[317] Maarten Keijzer. "Scaled symbolic regression". In: *Genetic Programming and Evolvable Machines* 5.3 (2004), pp. 259–269. DOI: 10.1023/B:GENP.0000030195.77571.f9.

[318] Marco Virgolin, Tanja Alderliesten, and Peter A. N. Bosman. "Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 1084–1092. ISBN: 9781450361118. DOI: 10.1145/3321707.3321758.

[319] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[320] Christopher J Moore, Alvin JK Chua, Christopher PL Berry, and Jonathan R Gair. "Fast methods for training Gaussian processes on large datasets". In: *Royal Society open science* 3.5 (2016), p. 160125. DOI: 10.1098/rsos.160125.

[321] Apostolos F Psaros, Xuhui Meng, Zongren Zou, Ling Guo, and George Em Karniadakis. "Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons". In: *Journal of Computational Physics* 477 (2023), p. 111902. DOI: 10.1016/j.jcp.2022.111902.

[322] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Icml*. 2010. DOI: 10.5555/3104322.3104425.

[323] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. "Improved training of wasserstein gans". In: *Advances in neural information processing systems* 30 (2017).

[324] Zijun Zhang. "Improved adam optimizer for deep neural networks". In: *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE. 2018, pp. 1–2. DOI: 10.1109/IWQoS.2018.8624183.

[325] Richard P Feynman, Robert B Leighton, and Matthew Sands. "The feynman lectures on physics; vol. i". In: *American Journal of Physics* 33.9 (1965), pp. 750–752. DOI: 10.1119/1.1972241.

[326] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. "Inference of regular expressions for text extraction from examples". In: *IEEE Transactions on Knowledge and Data Engineering* 28.5 (2016), pp. 1217–1230. DOI: 10.1109/TKDE.2016.2515587.

[327] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. DOI: 10.5555/1953048.2078195.

[328] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-

an-imperative-style-high-performance-deep-learning-library.pdf.

[329] Marco Virgolin. *genepro*. Version 0.1.0. Sept. 2022. URL: https://github.com/marcovirgolin/genepro.

[330] David Harrison Jr and Daniel L Rubinfeld. "Hedonic housing prices and the demand for clean air". In: *Journal of environmental economics and management* 5.1 (1978), pp. 81–102. DOI: 10.1016/0095-0696(78)90006-2.

[331] Athanasios Tsanas and Angeliki Xifara. "Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools". In: *Energy and buildings* 49 (2012), pp. 560–567. DOI: 10.1016/j.enbuild.2012.03.003.

[332] Athanasios Tsanas. "Accurate telemonitoring of Parkinson's disease symptom severity using nonlinear speech signal processing and statistical machine learning". PhD thesis. Oxford University, UK, 2012.

[333] Charles Spearman. "Footrule for measuring correlation". In: *British Journal of Psychology* 2.1 (1906), p. 89.

[334] Persi Diaconis and Ronald L Graham. "Spearman's footrule as a measure of disarray". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.2 (1977), pp. 262–268. DOI: 10.1111/j.2517-6161.1977.tb01624.x.

[335] Eckart Zitzler and Simon Künzli. "Indicator-based selection in multiobjective search". In: *International conference on parallel problem solving from nature*. Springer. 2004, pp. 832–842. DOI: 10.1007/978-3-540-30217-9_84.

[336] Lorenzo Bonin, Luigi Rovito, Andrea De Lorenzo, and Luca Manzoni. "Cellular geometric semantic genetic programming". In: *Genetic Programming and Evolvable Machines* 25.1 (2024), pp. 1–32. DOI: 10.1007/s10710-024-09480-8.

[337] Matthew Streeter and Lee A Becker. "Automated discovery of numerical approximation formulae via genetic programming". In: *Genetic Programming and Evolvable Machines* 4 (2003), pp. 255–286. DOI: 10.1023/A:1025176407779.

[338] Thomas F Brooks, D Stuart Pope, and Michael A Marcolini. *Airfoil self-noise and prediction*. Tech. rep. 1989.

[339] I-C Yeh. "Simulation of concrete slump using neural networks". In: *Proceedings of the Institution of Civil Engineers-Construction Materials* 162.1 (2009), pp. 11–18. DOI: 10.1680/coma.2009.162.1.11.

[340] Francesco Archetti, Stefano Lanzeni, Enza Messina, and Leonardo Vanneschi. "Genetic programming for computational pharmacokinetics in drug discovery and development". In: *Genetic Programming and Evolvable Machines* 8 (2007), pp. 413–432. DOI: 10.1007/s10710-007-9040-z.

[341] I Ortigosa, R Lopez, and J Garcia. "A neural networks approach to residuary resistance of sailing yachts prediction". In: *Proceedings of the international conference on marine engineering MARINE*. Vol. 2007. 2007, p. 250.

[342] Patrick AP Moran. "Notes on continuous stochastic phenomena". In: *Biometrika* 37.1/2 (1950), pp. 17–23. DOI: 10.2307/2332142.

[343] Hongfei Li, Catherine A Calder, and Noel Cressie. "Beyond Moran's I: testing for spatial dependence based on the spatial autoregressive model". In: *Geographical analysis* 39.4 (2007), pp. 357–375. DOI: 10.1111/j.1538-4632.2007.00708.x.

[344] Luc Anselin and Serge Rey. "Properties of tests for spatial dependence in linear regression models". In: *Geographical analysis* 23.2 (1991), pp. 112–131. DOI: 10.1111/j.1538-4632.1991.tb00228.x.

[345] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Vol. 2. Cambridge university press, 2001. DOI: 10.5555/1804390.

[346] Shafi Goldwasser and Silvio Micali. "Probabilistic encryption & how to play mental poker keeping secret all partial information". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. STOC '82. San Francisco, California, USA: Association for Computing Machinery, 1982, pp. 365–377. ISBN: 0897910702. DOI: 10.1145/800070.802212.

[347] Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. "Genetic programming: An introductory tutorial and a survey of techniques and applications". In: *Univ. Essex School of Computer Science and Eletronic Engineering Technical Report No. CES-475* (2007), pp. 1–112.

[348] Hans-Dieter Block. "The perceptron: A model for brain functioning." In: *Reviews of Modern Physics* 34.1 (1962), p. 123. DOI: 10.1103/RevModPhys.34.123.

[349] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.