FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Co-evolutionary approach to symbolic regression
Student:	Přemysl Pilař
Supervisor:	Ing. Erik Derner, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Symbolic regression (SR) is a machine-learning technique used to fit a set of data samples with a model in the form of an analytic expression. The expression is composed of constants, variables, and user-defined elementary functions such as {+, -, *, /, sin, log}. Typically, the SR task is solved using genetic programming (GP). GP is a gradient-free evolutionary method that iteratively evolves expressions by means of genetic operators while minimizing the objective function, e.g., the mean-square error. It has been shown that GP-based approaches work best if the model is represented as a linear combination of a set of possibly nonlinear features.

The goal of this work is to design a co-evolutionary algorithm that simultaneously evolves a diverse set of useful features and another population of well-performing models composed of the features from the first population.

The objective of this work can be decomposed into the following sub-tasks:
1. Choose a suitable symbolic regression method that facilitates the implementation of the co-evolution of features and candidate solutions.
2. Design and implement co-evolution for symbolic regression, where one population represents features and the individuals in the other one are linear combinations of the features.
3. Explore the possible ways of measuring the quality (fitness) of the features.
4. Suggest a technique to maintain the diversity in both populations, avoiding a premature convergence to just a few sub-optimal solutions.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 10 May 2022 in Prague.

Bachelor's thesis

CO-EVOLUTIONARY APPROACH TO SYMBOLIC REGRESSION

Přemysl Pilař

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Erik Derner, Ph.D.
June 29, 2023

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Pilař Přemysl. *Co-evolutionary approach to symbolic regression* . Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

List of Tables

List of code listings

Acknowledgement

I would like to thank my supervisor Ing. Erik Derner, Ph.D., for his valuable time. I would also like to thank Ing. Jiří Kubalík Ph.D. and prof. Dr. Ing. Robert Babuška for allowing me to work on the project with them.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on June 29, 2023 .....

Abstract

Symbolic regression is a stochastic machine learning method which tries to find the best model for given data composed of elementary mathematical functions. It develops its models using genetic operators and evolutionary techniques. This thesis is devoted to the co-evolution of features and models inside a symbolic regression, specifically Single Node Genetic Programming. It serves as a proof of concept of this method demonstrated on problems like parallel resistors or magnetic manipulation. The proof is done on the single objective optimisation

Keywords Symbolic Regression, Single Node Genetic Programming, Co-evolution, Single Objective Optimisation, Genetic Programming, Model Learning

Abstrakt

Symbolická regrese je stochastická metoda strojového učení. která se pokouší hledat nejlepší model pro daná data, skládající základní matematické funkce. Své modely vylepšuje pomocí genetických operátorů a evolučních technik. Tato práce je věnována koevoluci příznaků a modelů uvnitř symbolické regrese, respektive Single Node Genetic Programming. Slouží jako důkaz funkčnosti této metodu testovanéé na problémech jako je zapojení paralelních rezistorů či manipulace s magnetem. Tento důkaz je uskutečňován na jednocílové optimalizaci

Klíčová slova symbolická regrese, SNGP, koevoluce, jednocílová optimalizace, genetické programování, učení modelu

list of Abbreviations

GP	Genetic Programming
MSE	Mean Square Error
SR	Symbolic Regression
SNGP	Single Node Genetic Programming

Chapter 1
Introduction

Symbolic regression is a technique for determining a suitable mathematical model to describe observed data [1]. In traditional regression modelling, the process starts with parameters optimized for a specific model, such as the relationship between input data and examined variables in linear regression or neuron structure in neural networks.

However, no such assumptions regarding the precise form of the function are necessary for symbolic regression. Instead, candidate function building blocks such as mathematical operators, variables, constants, and analytic functions are provided, and symbolic regression then searches through the space spanned by these basic building blocks to find the best solution. In other words, in symbolic regression, both model structures and model parameters are optimized. The optimisation algorithms used in symbolic regression differ from traditional analytical/numerical optimisation methods because there is no requirement for a predefined function form [2].

Chapter 2
Context of the thesis

2.1 Problem definition

The Single Node Genetic Programming (SNGP) population is a set $M = m_0, m1, \ldots, m_L - 1$ of L individuals, with each individual being a single node represented by a tuple $m_i = <e_i, Succ_i, Pred_i, o_i, f_i>$ [3] where

- $e_i \in T \bigcup F$ is element from function set F or a terminal set T;
- $Succ_i$ is a set of successors of this node, the nodes that serve as input to this one;
- $Pred_i$ is a set of predecessors of this node, the nodes that use this node as input
- o_i is a vector of outputs
- f_i is the fitness

The architecture was expanded to a set of M independent models where each model is a base SNGP population as described above [4].

2.1.1 Original SNGP implementation

The original SNGP implementation that was provided to me comes from these articles [3, 4, 5]. Its single population consists of constants, variables, function nodes chosen from a set of elementary functions, and identity nodes, which are connected in a left-right manner. That means the node can be input (successor) only to those on the right (predecessors) of it in the population and vice-versa. Each node is a root of an expression, and it can be traversed recursively to terminal leaf nodes in the graph [3].

Every node keeps its own output calculated based on the training data. Not all nodes have a variable as input data, and its output is constant for these conditions easing up the calculation process. Even though we call it output, its relation with model fitness is complex.

Changes must be made for the population's ability to progress and fit better into a given dataset. A simple successor mutation (smut) was implemented between each generation. It chooses a random node of the population, chooses one of its successors and then finds a random node on the left to replace the chosen successor [3].

Identity nodes represent the actual model. Each identity node has one successor, the link to another node in the population, and its output is simply the output of the linked node. Given the set of these nodes, we are presented with a model which can be evaluated. The model $f(x)$ follows this formula

$$f(x) = \beta_0 + \sum_{i=1}^{n_f} \beta_i \varphi_i(x) \tag{2.1}$$

where φ_i are features/expressions evolved through the means of the smut. These features are non-constant combinations of nodes originating from the population. The coefficients β_i are not part of the population and do not undertake any mutations. Instead, they are estimated using the least squares technique [4]. Two user-defined parameters keep model complexity in check. n_f is the maximum number of identity nodes in one population. The second parameter is the maximum depth of any given feature [4].

At last, these populations (SNGPs) and their models create another population. The population of multiple SNGPs allows population-based methods [4]. Every SNGP has fitness in the form of MSE and, therefore, can be compared to others. This could not be done on a node scale since nodes need to indicate their performance adequately.

2.2 Challenges in Finding Formulas for Models

2.2.1 Search Space Exploration

Search space exploration is integral to symbolic regression since it explores a considerable universe of alternative formulas to discover the best representation for the underlying connections in the provided data. Yet, owing to the innate complexity and vastness of the search universe, this exploration method creates substantial problems [6].

Searching through a space that contains numerous combinations and arrangements of mathematical operations, variables, and constants is known as symbolic regression. The search space expands exponentially as the number of variables and possible mathematical operations rises, making comprehensive exploration computationally intensive [7].

Researchers have developed several methodologies and approaches for practical search space exploration in symbolic regression to solve this difficulty. These algorithms try to strike a compromise between exploration and exploitation, guaranteeing complete search space exploration while discovering attractive locations effectively.

Heuristic search strategies are a prevalent tactic. To develop and enhance candidate formulas iteratively, these approaches include genetic programming, genetic algorithms, particle swarm optimisation, or ant colony optimisation. These algorithms explore the search space for high-quality answers by using genetic operators such as mutation and crossover [6].

Another way is to use sampling or subset selection methods. Instead of scanning the whole search space, these approaches analyse and enhance a representative selection of formulas. Random sampling, Latin hypercube sampling, or Monte Carlo sampling methods may be utilised to examine a wide range of formulas effectively. Subset selection strategies reduce the dimensionality of the search space by identifying a smaller subset of variables or mathematical operations that are most useful or relevant to the issue [6].

Multi-objective optimisation techniques [8] have also been used to solve the difficulty of search space exploration. These algorithms investigate several parts of the search space at the same time, taking into account numerous competing goals such as accuracy, model complexity, and interpretability. These algorithms allow the identification of solutions that balance opposing goals by examining the Pareto front, representing a spectrum of trade-off solutions.

Integrating past information and limits may help steer search space research even further. Using domain-specific information or setting limits on the search process, researchers might steer the search towards more promising locations. This may assist in increasing the exploration process's efficiency and efficacy.

Effective symbolic regression search space exploration requires careful consideration of the trade-offs between exploration and exploitation and the efficient use of computer resources. Researchers want to lower the computational cost, enhance search efficacy, and raise the possibility of discovering correct and interpretable formulas by utilising appropriate search tactics and methodologies[9].

2.2.2 Overfitting and Underfitting

Overfitting and underfitting are common challenges in machine learning, including symbolic regression. They refer to the phenomena where a model fails to generalise well to unseen data due to excessive complexity or insufficient flexibility, respectively.

Overfitting occurs when a model becomes overly complex and too closely fits the training data, capturing noise and irrelevant patterns instead of the underlying relationships. This leads to poor generalisation, where the model performs well on the training data but poorly on new, unseen data. Overfitting can occur when a model has too many variables, incorporates too many features, or is too complex. It often results from an overly ambitious attempt to capture all the details of the training data, failing to discern proper underlying patterns [10].

On the other hand, underfitting arises when a model is too simplistic or lacks the necessary flexibility to capture the underlying relationships in the data. An underfit model has not learned enough from the training data and struggles to capture the complexity of the problem. It typically leads to poor performance in the training and unseen data, as it fails to capture the essential patterns and trends [10].

To address overfitting and underfitting in symbolic regression, various techniques can be employed:

1. **Regularisation:** Regularization techniques introduce additional constraints on the model during training to prevent overfitting. Standard regularisation methods include L1 and L2 regularisation, which add penalty terms to the objective function, discouraging the model from relying too heavily on specific variables or features [11].

2. **Cross-validation:** Cross-validation helps assess the model's generalisation performance by splitting the available data into training and validation sets. This enables evaluating the model's performance on unseen data and helps identify overfitting or underfitting. Techniques like k-fold cross-validation or hold-out validation can be employed to obtain more reliable estimates of the model's performance [12].

3. **Feature selection and dimensionality reduction:** Selecting relevant features and reducing the dimensionality of the input space can mitigate overfitting. By focusing on informative

features, the model can avoid noise and irrelevant patterns in the data. Techniques like forward selection, backward elimination, or principal component analysis (PCA) can be used for feature selection and dimensionality reduction [13].

4. **Early stopping:** Early stopping involves monitoring the model's performance on a validation set during training and stopping the training process when the performance degrades. This prevents the model from excessively fitting the training data and helps find a balance between model complexity and generalisation [14].

5. **Ensemble methods:** Ensemble methods combine multiple models to improve predictive performance and mitigate overfitting. Techniques such as bagging, boosting, or random forests can aggregate the predictions of various models and reduce the impact of individual model biases or errors [14, 15].

Researchers aim to find a model that generalises to unseen data and captures the genuine underlying relationships by addressing overfitting and underfitting in symbolic regression. These techniques contribute to developing accurate and reliable models that can be effectively applied in various domains.

2.2.3 Computational Efficiency

Computational efficiency is essential in symbolic regression as it directly impacts the practicality and scalability of the technique. Symbolic regression involves exploring a large search space of possible formulas, which can be computationally demanding. Therefore, improving computational efficiency is essential to make symbolic regression feasible for real-world applications.

There are several strategies to enhance computational efficiency in symbolic regression:

1. **Optimized search algorithms:** Efficient search algorithms are crucial for navigating the search space effectively. Genetic programming techniques can be optimised by implementing advanced genetic operators, such as subtree crossover and mutation, designed explicitly for symbolic regression. These operators should be carefully crafted to balance exploration and exploitation, ensuring a thorough search without excessive computational overhead [16].

2. **Parallel computing:** Symbolic regression algorithms can benefit from parallel computing techniques to distribute computations across multiple processors or machines. Parallelisation can significantly reduce the execution time, particularly when evaluating the fitness of candidate solutions, performing crossover and mutation operations, or conducting extensive simulations. Utilising multi-core processors, clusters, or GPU acceleration can exploit parallelism and boost computational efficiency [17].

3. **Data preprocessing and feature selection:** Preprocessing the data and selecting informative features can reduce the computational burden in symbolic regression. Data cleaning, dimensionality reduction techniques, and feature selection methods can help eliminate noise, reduce the problem's dimensionality, and focus the search on relevant features [18].

4. **Model complexity control:** Techniques such as regularisation or complexity penalties can prevent the model from becoming overly complex, reducing the risk of overfitting and improving computational efficiency during training and evaluation. Encouraging sparsity in the evolved models can also simplify calculations and make the models more interpretable [19].

5. **Sampling and approximation:** Rather than exhaustively evaluating all possible formulas, sampling techniques and approximation methods can be employed. These approaches

involve assessing a subset of candidate solutions or using surrogate models to estimate the fitness values. Sampling and approximation reduce the computational burden while providing reasonable solutions, mainly when dealing with large datasets or complex formulas [20].

6. **Algorithmic optimizations:** Algorithmic optimisations specific to symbolic regression can enhance computational efficiency. For example, memoisation techniques can store and reuse intermediate calculations, avoiding redundant computations. Additionally, carefully selecting evaluation strategies and data structures, such as expression trees or matrix representations, can improve efficiency during fitness evaluation and model manipulation [21].

Efficient computational techniques in symbolic regression are crucial to handle the search space's complexity. By optimising search algorithms, using parallel computing, preprocessing data, controlling model complexity, employing sampling and approximation, and implementing algorithmic optimisations, researchers can significantly enhance the computational efficiency of symbolic regression algorithms, enabling their practical application to large-scale and real-world problems.

2.2.4 Interpretability and Model Complexity

Interpretability refers to understanding and explaining how a model arrives at its predictions. In symbolic regression, interpretability is achieved by representing the models as human-readable formulas composed of mathematical expressions, variables, and elementary functions. Symbolic regression excels in this field, unlike its competitors. Interpretable models are desirable as they allow domain experts to gain insights, validate the model's behaviour, and make informed decisions based on the learned formulas. By providing transparent and understandable representations, symbolic regression models can facilitate the discovery of meaningful patterns and relationships in the data [22].

Model complexity refers to the sophistication of the learned formulas. While a complex model may have a higher potential to fit the training data accurately, it runs the risk of overfitting, making it difficult to generalise to new, unseen data. On the other hand, an overly simplistic model may underfit, failing to capture the complexity of the underlying relationships. Balancing between model complexity and performance is essential to ensure accurate predictions while maintaining interpretability.

1. **Regularization:** Regularisation techniques, such as L1 or L2 regularisation, can be employed to control model complexity. By adding penalty terms to the objective function, regularisation discourages the model from relying too heavily on certain variables or features, promoting sparsity and reducing unnecessary complexity [11].

2. **Feature selection:** Careful feature selection can simplify the model and improve interpretability. Identifying and selecting relevant features that capture the essential aspects of the problem can help reduce noise, eliminate irrelevant variables, and create more interpretable formulas [13].

3. **Simplification techniques:** After obtaining a symbolic regression model, simplification techniques can be applied to reduce complexity without sacrificing accuracy. Simplification involves eliminating redundant terms, combining similar terms, or applying mathematical identities to create more concise and interpretable formulas. Techniques such as genetic programming-based simplification or symbolic algebraic manipulation can be employed [23].

4. **Trade-off analysis:** Researchers need to find an optimal trade-off between model complexity and performance by evaluating different models with varying levels of complexity. Techniques

such as cross-validation can help assess the generalisation performance of the models and guide the selection of the best compromise between accuracy and interpretability [24].

5. **Visualization and explanation:** Developing visualisation techniques and tools can aid in interpreting and explaining the learned formulas. Graphical representations, such as plots or diagrams, can provide intuitive insights into the relationships between variables and the overall behaviour of the model. Additionally, providing explanations or justifications for the model's decisions and predictions can enhance interpretability and build trust in the model [25].

Symbolic regression models can generate accurate and understandable formulas for domain experts by considering interpretability and managing model complexity. The ability to interpret the learned formulas empowers researchers and practitioners to gain valuable insights, validate the model's behaviour, and effectively apply symbolic regression in various fields where interpretability is essential.

2.2.5 Maintaining Diversity

Without any techniques to maintain diversity, populations could quickly converge to a few sub-optimal solutions in the local optimum. Therefore it is yet another key aspect to consider when designing a formula-finding algorithm [26].

1. **Population Initialisation:** Proper population initialisation is the first step in maintaining diversity. It is important to generate an initial population of candidate formulas that covers a diverse range of solutions. This can be achieved by employing various techniques, such as random initialisation or using diverse seed formulas based on prior knowledge or domain expertise [27].

2. **Selection Operators:** The selection process in evolutionary algorithms determines which individuals are chosen to reproduce and produce offspring for the next generation. Different selection strategies can influence the diversity of the population. For example, tournament selection or roulette selection methods can preserve diversity by ensuring that a wide range of individuals has the chance to be selected for reproduction [28].

3. **Genetic Operators:** Genetic operators, including mutation and crossover, create new candidate solutions in the population. When designing these operators, it is important to balance exploration and exploitation. Exploration encourages generating diverse solutions by introducing random variations, while exploitation focuses on refining promising solutions. Well-designed genetic operators can help maintain diversity by preserving a mix of both exploratory and exploitative solutions [29].

4. **Fitness Sharing:** Fitness sharing is a technique that promotes diversity by adjusting the fitness of individuals based on their similarities to others in the population. It introduces a penalty or reduction factor for individuals too similar to others, thereby encouraging the preservation of different solutions. By incorporating fitness sharing, the algorithm incentivises the exploration of different regions of the solution space, preventing the population from becoming dominated by a few highly fit individuals [30].

5. **Niching Methods:** Niching methods are designed to maintain diversity by partitioning the population into distinct niches or subgroups. These methods aim to allocate individuals to different niches based on their similarities and prevent overcrowding of solutions in any single niche. Niching methods allow the exploration of diverse regions of the solution space, enabling the algorithm to uncover a broader range of potential formulas [31].

2.3 Approaches for Formula Discovery

The discipline of formula discovery can be broad, and every small difference in the problem gave birth to a new method.

2.3.1 Statistical Regression Models

Statistical regression models are widely used for formula discovery and have a long history in data analysis. They aim to establish a relationship between a dependent variable and one or more independent variables by fitting a mathematical equation to the data. The most common types of statistical regression models include linear regression, polynomial regression, and support vector regression.

1. **Linear Regression:** Linear regression is a popular and straightforward statistical model that assumes a linear relationship between the independent and dependent variables. It seeks to find the best-fit line that minimises the sum of squared residuals. The resulting formula is a linear equation: y represents the dependent variable, x_1, x_2, ..., x_n are the independent variables, and b_0, b_1, b_2, ..., b_n are the estimated coefficients[7].

$$y = b_0 + \sum_{i=1}^{n} b_i x_i \tag{2.2}$$

   Linear regression models provide interpretable formulas that quantify the relationships between the variables. The estimated coefficients reveal the direction and magnitude of the effects of the independent variables on the dependent variable. Additionally, statistical inference techniques, such as hypothesis testing and confidence intervals, can be applied to assess the significance of the coefficients and the overall model.

2. **Polynomial Regression:** Polynomial regression extends linear regression by introducing polynomial terms of the independent variables. This allows for capturing non-linear relationships between the variables. The resulting formula becomes a polynomial equation by including higher-order polynomial terms (e.g., quadratic, cubic). Polynomial regression can approximate more complex data patterns and provide a better fit when the relationship between variables is non-linear[32].

   Polynomial regression models offer flexibility in capturing curvilinear trends and interactions between variables. However, as the degree of the polynomial increases, the model can become more prone to overfitting, leading to complex formulas that may not generalise well to new data.

3. **Support Vector Regression:** Support Vector Regression (SVR) is a machine learning technique that combines elements of statistical regression and support vector machines[33]. SVR aims to find a hyperplane that maximises the margin around the training data while satisfying a specified tolerance level, or "epsilon." The resulting formula can take different forms depending on the kernel function, such as linear, polynomial, or radial basis functions.

   SVR models provide flexibility in handling non-linear relationships using kernel functions. The formulas derived from SVR capture complex patterns in the data and offer accurate predictions. However, interpreting the formulas directly can be challenging due to the nature of the support vector machine framework.

4. **Symbolic Regression:** Symbolic regression is a machine learning technique to discover mathematical formulas or expressions that accurately represent the relationships within a given dataset. Unlike traditional regression methods that rely on predefined functional forms,

symbolic regression aims to automatically search and evolve mathematical expressions that best fit the data[31].

It explores a vast search space of possible formulas, incorporating variables, constants, and elementary functions to find the most suitable representation. Symbolic regression provides a powerful tool for modelling and understanding complex phenomena, allowing researchers and practitioners to uncover interpretable and insightful mathematical relationships from empirical data.

In summary, statistical regression models provide interpretable formulas representing variables' relationships. Linear regression offers simplicity and transparency, while polynomial and support vector regression can capture more complex non-linear patterns. These models allow researchers to discover formulas that can be easily understood and provide insights into the underlying mechanisms driving the data.

2.3.2 Neural Networks

Neural networks have emerged as powerful models for various data-driven tasks, including formula discovery. Inspired by the structure and functioning of the human brain, neural networks are composed of interconnected nodes, or "neurons," organised into layers. Each neuron receives input signals, processes them through an activation function, and produces an output signal. The strength of the connections between neurons, represented by weights, determines the impact of each neuron's input on the final output[34].

Neural networks capture complex relationships and patterns in data, making them suitable for formula discovery tasks. They can learn linear and non-linear relationships between variables, allowing them to uncover intricate mathematical formulas governing the data.

The architecture of a neural network is the main component. Deep neural networks with multiple hidden layers effectively learn complex representations and capture non-linear interactions between variables. The hidden layers enable the network to automatically extract relevant features from the input data, reducing the need for manual feature engineering.

During training, neural networks adjust their weights using optimisation algorithms such as gradient descent. By minimising a specified loss function, neural networks iteratively update the weights to improve their predictions and accurately fit the data. The chosen loss function depends on the specific formula discovery task, such as mean squared error for regression problems.

Interpreting the resulting formulas from neural networks can be challenging due to their inherent complexity and black-box nature. While the network's weights represent the learned relationships between variables, they are not easily translatable into explicit mathematical expressions. However, lately, heterogenous neural networks are being developed where each node in a layer is an elementary function[34].

Researchers have developed methods such as heterogeneous neural networks to improve the interpretability of neural network models. These approaches aim to combine the power of neural networks in capturing complex patterns with the interpretability of symbolic regression, resulting in accurate and human-readable formulas.

Neural networks offer great flexibility and the ability to discover formulas in a data-driven manner. Their ability to handle large-scale and high-dimensional datasets and their capacity to model complex relationships make them valuable tools for formula discovery in various domains.

2.3.3 Bayesian Methods

Bayesian methods provide a principled and probabilistic framework for formula discovery. These methods are rooted in Bayesian inference, which involves updating prior beliefs based on observed data to obtain posterior probability distributions. In the context of formula discovery, Bayesian methods allow for estimating uncertainty and provide a flexible approach to model selection and parameter estimation[35].

One of the major advantages of Bayesian methods is their ability to handle uncertainty explicitly. Bayesian models can quantify the confidence or uncertainty associated with the estimated formulas by representing uncertainty as probability distributions. This is particularly useful when dealing with limited data or complex and noisy phenomena.

Bayesian formula discovery typically involves specifying a prior distribution over possible formulas and updating it using observed data. This allows for a more robust and flexible approach than traditional frequentist methods. Bayesian methods can handle small sample sizes more effectively and incorporate prior knowledge or beliefs into the modelling process.

Markov chain Monte Carlo (MCMC) methods, such as the Metropolis-Hastings algorithm and Gibbs sampling, are commonly used in Bayesian formula discovery. These methods generate samples from the posterior distribution of the formulas, allowing for exploring the formula space and estimating model parameters. By sampling from the posterior distribution, Bayesian methods can provide point estimates and credible intervals that capture the uncertainty associated with the estimated formulas.

Another advantage of Bayesian methods is their ability to perform model selection. Through Bayesian model selection, different candidate formulas or models can be compared based on their posterior probabilities, considering the goodness of fit and model complexity. This approach helps avoid overfitting and provides a principled way to choose the most suitable formula for a given dataset.

Bayesian methods in formula discovery can also incorporate hierarchical structures, where formulas at different levels of the hierarchy capture different aspects of the phenomenon being modelled. This allows for integrating domain knowledge or prior information, providing a more structured and interpretable approach to formula discovery.

Interpreting the resulting formulas from Bayesian methods is often straightforward since the formulas are represented explicitly as mathematical expressions. The uncertainty estimates associated with the formulas provide insights into the reliability of the estimates, allowing for informed decision-making.

Overall, Bayesian methods offer a robust framework for formula discovery by providing a probabilistic approach that incorporates uncertainty, performs model selection and allows for the integration of prior knowledge. They are particularly useful in scenarios where uncertainty quantification, small sample sizes, or the incorporation of prior beliefs are important considerations.

2.4 Fitness Evaluation and Objective Functions

2.4.1 Fitness Evaluation Metrics

Evaluating the fitness of candidate solutions relies on metrics that provide insights into the performance and quality of the evolved formulas. These metrics guide the evolutionary process and help us understand how well the generated solutions fit the data.

One commonly used metric is the mean square error (MSE), which measures the average squared difference between the predicted values of the evolved formula and the actual observed values from the data. The MSE gives us an idea of how well the formula fits the data overall[31].

For formulas bellow y_i denotes observed value of i-th sample, $\hat{y}_i$ denotes predicted value of i-th sample and n represents number of samples

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

To complement the MSE, we often consider the root mean square error (RMSE), which is obtained by taking the square root of the MSE. The RMSE provides a more interpretable measure of the average size of the errors.

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

Another helpful metric is the mean absolute error (MAE), which calculates the average absolute difference between the predicted values and the actual values. The MAE provides a straightforward measure of the average magnitude of the errors without squaring them. It can offer a more intuitive understanding of the absolute accuracy of the evolved formula. It is important to mention that this metric differs from the other two. It is less prone to outliers as its error is not squared. This might be a desired quality for certain problems.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Utilising these metrics, including MSE, RMSE, and MAE, we can assess different aspects of the performance of the evolved formulas. The MSE and RMSE help us understand how well the formulas fit the data regarding squared errors, while the MAE provides insight into the average absolute deviation from the actual values. Examining these metrics allows us to compare and evaluate different solutions and make informed decisions in the symbolic regression process.

It's important to note that these fitness evaluation metrics are not the only considerations in symbolic regression. Other metrics, such as R-squared and complexity-based measures, also provide valuable insights into the quality and interpretability of the evolved formulas. By considering multiple metrics, we can gain a comprehensive understanding of the fitness of the formulas and make well-informed choices in the evolutionary search process.

2.4.2 Testing techniques

1. **Cross-Validation:** Cross-validation is a widely used technique in machine learning for evaluating the performance of models. It involves dividing the dataset into multiple subsets or folds, typically with a predefined ratio, such as k-fold cross-validation. Each fold is used as a validation set, while the remaining folds are used for training. This process is repeated

k times, with each fold being the validation set once. By averaging the performance across all folds, cross-validation provides a robust estimate of the model's performance on unseen data.

The benefit of cross-validation lies in its ability to assess how well a model generalises to different data samples. It helps mitigate the risk of overfitting, where a model performs well on the training data but fails to generalise to new instances. Cross-validation provides a more realistic evaluation of the model's capabilities by testing it on various subsets of the data, providing insights into its performance under different conditions.

A popular variation of cross-validation is k-fold cross-validation, where the dataset is divided into k equally sized folds. The model is trained on k-1 folds and validated on the remaining fold. This process is repeated k times, ensuring each fold is the validation set once. The performance metrics from each iteration are then averaged to obtain an overall estimate of the model's performance. K-fold cross-validation is commonly used when a larger dataset is available and provides a more robust assessment of the model's generalisation capabilities than traditional cross-validation.

2. **Holdout Testing:** Holdout testing is another standard method used for evaluating machine learning models. In holdout testing, a portion of the dataset is set aside and not used during training. Once the model has been trained on the remaining data, it is evaluated on the holdout set. This allows for an unbiased assessment of the model's performance on unseen data.

   The holdout set is a proxy for real-world data, indicating how the model will perform in practice. It helps identify potential issues such as overfitting or underfitting and measures the model's ability to generalise. However, it is crucial to ensure that the holdout set is representative of the overall data distribution to obtain reliable performance estimates.

3. **Bootstrapping:** Bootstrapping is a statistical resampling technique commonly used in machine learning testing. It involves randomly sampling the dataset with replacement to create multiple bootstrap samples. Each bootstrap sample is used to train a separate model, and the performance metrics are computed on each model. The variability in the performance metrics across the bootstrap samples estimates the uncertainty and robustness of the model's performance.

   Bootstrapping is particularly useful when the available dataset is limited. It allows for generating additional samples, mimicking the process of drawing new samples from the population. This helps to mitigate the impact of data scarcity and provides a more reliable estimate of the model's performance.

Chapter 3
Solution

3.1 Single Node Genetic Programming

SNGP is a graph-based GP technique that evolves a population organized as an ordered linear array of individuals, each representing a single program node [3].

Each node contains information about its placement in the pool of graphs and its type. In addition to the attributes encoding connections with other members, each individual has data structures recording the outputs it produces when evaluated; more importantly, it has its own distinct fitness value. It, therefore, makes much more sense to view the population as a set of graphs, with each individual holding the root node of an expression or program to be evaluated [36].

3.2 Co-evolution in SNGP

In this section, I would like to introduce the proposed changes and their implementation based on the assignment. To-be augmented implementation was described in the section 2.1.1

3.2.1 Inputs and outputs of SNGP

The central part of this thesis revolves around the research of a method that co-evolves models and their sub-parts called features. As mentioned above, a few simple steps exist to implement this idea.

The program uses many parameters which enable fine customisation of the evolution. Here are all parameters used in the augmented version of the algorithm. These arguments are passed in a configuration file; the most important ones can be passed through the command line.

- **runs -** number of times the program will run, incrementing the seed each time
- **seed -** starting seed
- **DEPTHLIMIT -** sets the maximum depth of an expression
- **data_file -** path to the training data
- **experiment -** type of problem to be solved (resistors, magman, pendulum, turtle bot)

- **out_file -** path to the output file
- **constants -** list of constants to be used
- **functions -** list of implemented elementary functions to be used
- **nbOfMutations -** number of feature mutations within one model
- **num_iterations_models -** number of mutations and crossovers done on models
- **model_size -** number of nodes in each SNGP population
- **pop_size -** number of SNGP populations and models
- **num_generations -** number of generations
- **tournament_tsize -** size of the tournament for tournament selection
- **max_duplicates -** number of models passed to the next generation originating from the same model
- **head_nodes -** number of constant nodes in each SNGP population
- **tail_nodes -** number of non-constant function nodes
- **identity_nodes -** number of identity nodes
- **p_head -** probability of mutating head node
- **p_mod_mutation -** probability of mutating the model
- **p_mod_crossover -** probability of performing model crossover

The output of the program is stored in a specified file. It has the following structure:

mse = mse

$f(x_0, ..., x_n) = \beta_0 + \beta_1 * I_1 + \ldots + \beta_i * I_i$

The output represents the best model in a given run. If $runs$ exceeds 1, multiple models are saved using the naming convention model_seed=$seed$.txt.

3.2.2 Separating features and models

Knowing what goes in and out of the program, we can continue with the implementation. To let two distinct populations evolve by themselves, they must be created. So far, there is only one population of SNGPs. The proposal is to take these SNGPs and separate identity nodes representing models and the parts with function, variable, and constant nodes.

Splitting these populations includes many changes to the current working program. It involves changing the structure of nodes and introducing coordinates to successors and predecessors compared to the previous simple index linking only inside one SNGP. This allows nodes to point through indices to the desired node correctly. Note that the feature node may only point to other nodes inside its own feature part of the former SNGP or to an identity node. These coordinates are combined with another attribute that deems if there if the pointed node is an identity node, thus needing additional coordinates. Conversely, the identity node may point to a feature outside the original model, thus requiring additional coordinates for its successor.

The population of sets of identity nodes had to be stored in a whole new class, giving birth to the *modelId* class. This class was designed to mirror as much as it could a previous SNGP population. Objective function calculation stayed as 2.1, and model construction remains unchanged.

What changed, however, were evolutionary techniques. Smut of identity nodes in this distinct population allows to node to alter its successor to a different feature population. A curious design choice is that if an identity node points to the constant node or a node with a constant value, it is not disregarded as an invalid successor. Instead, this identity node is not passed to the least squares method since it bears no meaning for another constant in the solution. Another addition is the crossover method. Implemented crossover method is a One-point crossover which chooses a random point in the vector and swaps the content on one side generating [9], contrary to the original algorithm, only one model is returned. Implementation does not allow empty crossover. If crossover should happen according to probability, it will yield a meaningful result.

One generation in *modelId* displays the 3.1 pattern.

Code listing 3.1 One generation of modelId evolution

```
for i in range(number_of_iterations):
    old_models = modelIds_list.copy()
    id = tournament_selection(modelIds_list)
    new_model = modelIds_list[id].evolve()
    new_model.calculate_output()
    new_model.calculate_fitness()
    new_model_list.append(new_model)
modelIds_list = merge(old_models, new_model_list)
relinkIds()
```

In one generation, numerous models are chosen based on the number of iterations through tournament selection. The tournament selection takes a subset of the tournament size and chooses the individual model with the best fitness [9]. This helps maintain diversity but more about that later. The new model is created with either crossover, smut or just passed down as it is. However, evolve method has an internal loop changing the model multiple times. Afterwards, outputs and coefficient β_i in equation 2.1 must be updated. The model is appended to the list of new potential models. The merge happens into a new population of models based on their objective function with a condition. More about it in the subsection. 3.2.4. It finishes by verifying all changed links.

3.2.3 Features evaluation and evolution

So far, apart from the crossover, there are no functional changes. To get the co-evolution running, a way to evaluate individual features must be designed and inserted into the framework. The proposal is to calculate the fitness of features derived from the models it is captured in. The next step is the modified evolution.

The fitness of the feature is calculated with aggregation function (median, average, minimum) of models it is contained in. A node not connected to any model is given a very high-value fitness. A change inside the features population will change the fitness of every feature connected to that given model and any other predecessors. Similarly, a change to a model will adjust the fitness of all connected features. As many changes are made each generation, calculating this value could significantly lower the speed of evolution. It is paramount to evaluate fitness only when needed for comparing fitness before and after mutation to accept or reject the proposal.

This pseudocode could denote one generation of feature mutation:

Code listing 3.2 One generation of feature evolution

```
iterations = generate_random_number(from = 0, to = nbOfMutations)
features_to_mutate_indices = create_random_permutation(iterations)
for i in features_to_mutate_indices:
    old_features = features_list[i].copy()
    new_features, mutated_node_index = features_list[i].mutate()
    new_features.evaluate_outputs
    features_list[i] = new_features
    models.update()
    models.feed_fitness_values()
    features_list[i].nodes[mutated_node_index].fitness_update()
    if old_features[mutated_node_index].fitness >=
       features_list[i].nodes[mutated_node_index]:
        features_list[i] = old_features
        models.update()
```

In each generation, a random number of iterations is generated up to $nbOfMutations$, and then indices for feature population to mutate are generated. A snapshot of the original feature population before mutation is taken for possible reversion of the change. Successor mutation is performed on a random node inside the population. The new feature population must evaluate its outputs first; afterwards, it replaces the old population. It then calculates the fitness of all affected model populations (identity node populations). Since models' fitness influences individual features' fitness, it is passed back to the feature populations. At last, fitness is updated within the node; if it yields better fitness, the population is kept as is; otherwise, rollback to snapshot takes place. I would like to point out that we keep calculations to a minimum, evaluating only needed values at any given time. Because of the frequency with which population changes, feature fitness is calculated each time as its only purpose is to accept or deny the proposed mutation.

3.2.4 Maintaining diversity

As already mentioned, there are a few mechanisms that stop premature convergence from happening. This algorithm's search space is smaller than other types of symbolic regressions. All nodes are created initially; the only changes during evolution are the successor replacement. This implies that no nodes are lost or made, only relinked. Combined with the maximum depth for expression and the maximum number of expressions included in the model, we have limited our search space to a subset of possible solutions. The first layer of maintaining diversity is that no nodes are ruled out of the population, and possible relink can effortlessly explore this part of space.

Mutations, even though usually exploitative [37], in the case of mutating a node not being used in a model, are accepted, eliminating the possibility that the node was already chosen by a model but rejected. Since no valid criterion is used to deem the mutation of a non-used node beneficial, it is accepted somewhat mindlessly.

The next layer comes with the implementation of crossover, which is viewed as an explorative operator [38]. The model to which genetic operators are applied is chosen using tournament selection which alleviates selection pressure [31] because the fittest individual is selected from a subset called the tournament.

When the merge happens, there lies another method. As we can spot at the start of subsection 3.2.1, there is a parameter called *max_duplicates*. This method ensures that only up to *max_duplicates* models will be selected, originating from the same one for the next generation. Initially, the fittest models are chosen, and if the threshold is reached, it will disregard any other model with that origin. Combined with the tournament selection, this already poses a significant diversity control.

3.2.5 Implementation of co-evolution

Described parts build the program. The main evolution loop looks like this:

Code listing 3.3 One generation of feature changes

```
for i in range(generations):
    new_models_list = []
    run_generation_features()  #Code 3.2
    run_generation_models()    #Code 3.1
    best_model_in_generation_list.append(best_model)
```

After defining the construction block, the code is simple. It clears the list for the new population. It evolves features since our main objective is to have a well-performing model. Feature evolution may not lead to better-performing models. This is the reason why models evolve second. Currently, there already are "better" features, and it could build models of better quality. Even though the building blocks might be better models, they may lead to worse ones. That is why keep a snapshot of the best model for each generation, as the final generation might not be the best.

3.2.6 Used technologies

Python was used to create the program code and test scripts. These libraries have been used:

- numpy for computations;
- matplotlib for visualisation;
- sklearn for metrics.

Chapter 4
Experiments

4.1 Tested problems

For these three problems, these parameters were used. Feature evaluation aggregation function was minimum.

- **Number of runs:** 50
- **Maximum feature's depth:** $n_f = 4$
- **Elementary Functions:** F = +,-,*,/,square,cube,sine,tanh
- **Population size:** $M = 50$
- **Tournament size:** 4
- **Number of generations:** 20
- **Number of identity nodes:** 10
- **Maximum number of duplicates:** 2
- **Model size:** 500

4.1.1 Parallel resistors

The first test problem is deriving a formula for the total resistance of parallel resistors in the circuit.

Which is

$$R_{\text{total}} = \frac{R_1 \cdot R_2}{R_1 + R_2} \tag{4.1}$$

This experiment aims to find a good approximation of this model. R_1 and R_2 are generated uniformly from the interval [0.0001, 20] Ω (Ohm). The variables and the target values are disturbed with up to 0.005 noise [4]. The training dataset has 200 records, and testing has 100 records.

Analysing the figure and table results, we can see that our method was success. Observing the maximum value, we can see that at least one model is overfitted. As expected, train MSE is slightly lower than test MSE. This difference is acceptable.

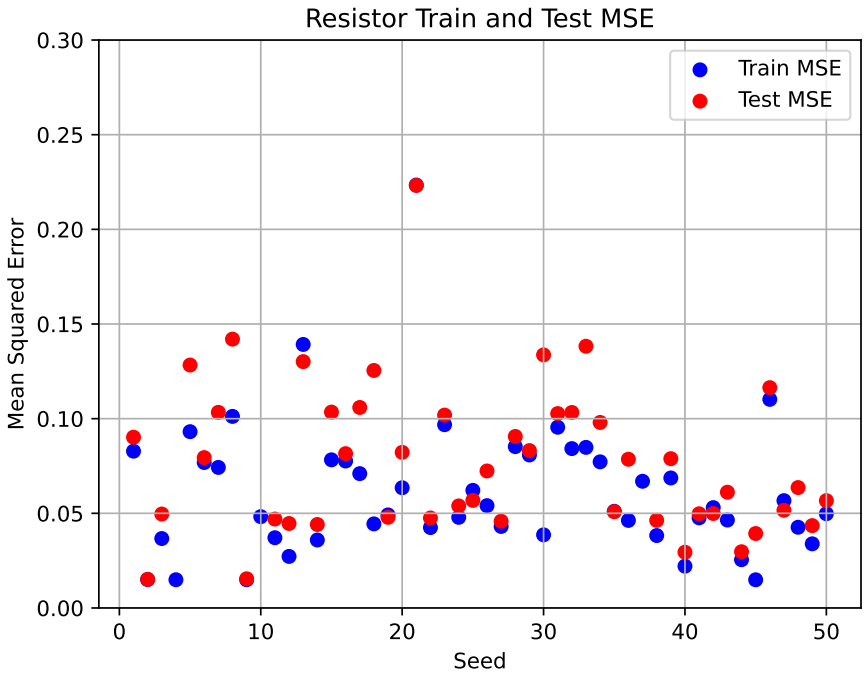

Figure 4.1 Resistor Train and Test MSE

Table 4.1 MSE Resistor Results

	Train MSE	Test MSE
Mean	0.061434	1.520110
Median	0.052040	0.078714
Standard Deviation	0.035773	8.183154
Minimum	0.014890	0.015126
Maximum	0.223400	57.281279

4.1.2 Magnetic Manipulation

An electromagnet is static beneath the rail, and an iron ball is in motion along the rail in the magnetic manipulation system. The objective is to find a model of the nonlinear magnetic force acting on the ball [4]. $f(x)$, as a function of the horizontal distance, x, between the iron ball and the activated coil, given a constant current through the coil, i. Data is generated with the following formula [39]:

$$\widetilde{f}(x) = -\frac{ic_1 x}{(x^2 + c_2)^3} \tag{4.2}$$

Training data are sampled from the interval [-0.075,0.075] m, and again both are disturbed by noise.
According to the figure and table, the method yielded good results. There were no outliers inside the best models of each run. What is strange is better performance on the test data than on train data. If we look at figure 4.5 we can see similar curves.

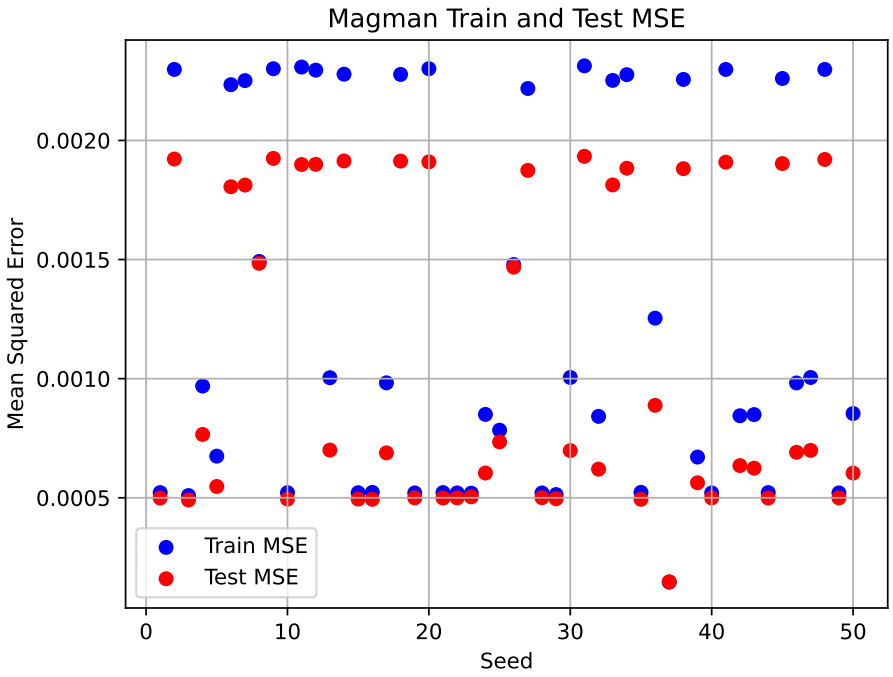

Figure 4.2 Magman Train and Test MSE

Table 4.2 MSE Magman Results

	Train MSE	Test MSE
Mean	0.001264	0.001055
Median	0.000975	0.000694
Standard Deviation	0.000774	0.000639
Minimum	0.000146	0.000148
Maximum	0.002313	0.001933

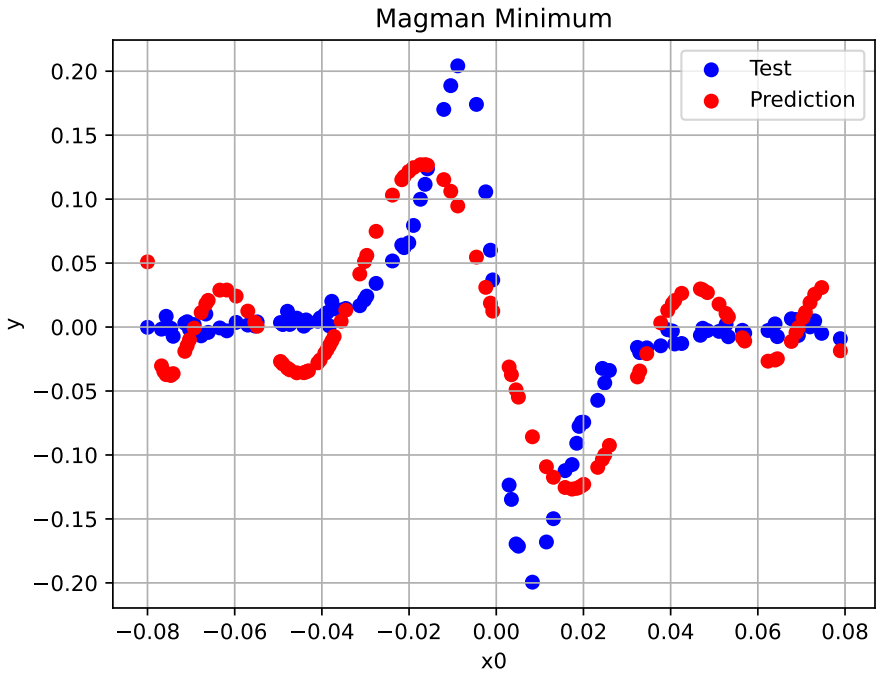

Figure 4.3 Magman plot with a minimum as fitness evaluation function

Table 4.3 MSE Turtle Bot Results

	Train MSE	Test MSE
Mean	0.000044	32.049991
Median	0.000039	0.000040
Standard Deviation	0.000020	226.627355
Minimum	0.000015	0.000014
Maximum	0.000106	1602.497436

4.1.3 Turtle Bot

A turtle bot is a small two-wheeled robot. Its state is described by a vector $\mathbf{x} = (x_{\text{pos}}, y_{\text{pos}}, \phi)$ where x_{pos} and y_{pos} are the position coordinates and ϕ is the robot's direction. The control input is $\mathbf{u} = (v_f, v_a)^T$ where v_f and v_a are desired forward and angular velocity. Models for the three state variables have the following form [40].

$$x_{\text{pos,k+1}} = f^{x_{\text{pos}}}(x_{\text{pos,k}}, y_{\text{pos,k}}, \phi_k, v_{\text{f,k}}, v_{\text{a,k}}), \tag{4.3}$$

$$y_{\text{pos,k+1}} = f^{y_{\text{pos}}}(x_{\text{pos,k}}, y_{\text{pos,k}}, \phi_k, v_{\text{f,k}}, v_{\text{a,k}}), \tag{4.4}$$

$$\phi_{\text{k+1}} = f^{\phi_{\text{pos}}}(x_{\text{pos,k}}, y_{\text{pos,k}}, \phi_k, v_{\text{f,k}}, v_{\text{a,k}}). \tag{4.5}$$

Experimental data were collected during the operation of the real robot. Five sequences of samples starting from the initial state $x_0 = (0,0,0)^T$ were generated with a sampling period T_s = 0.2 s. In each sequence, the robot was steered by 30 different pairs of random inputs v_f, v_a, keeping each pair of inputs constant for 5 samples. This yielded 150 samples per sequence. The random inputs were drawn from the domain $v_f \in$ [0,0.3] $m \cdot s^{-1}$, $v_a \in$ [-1,1] $rad \cdot s^{-1}$ [40].

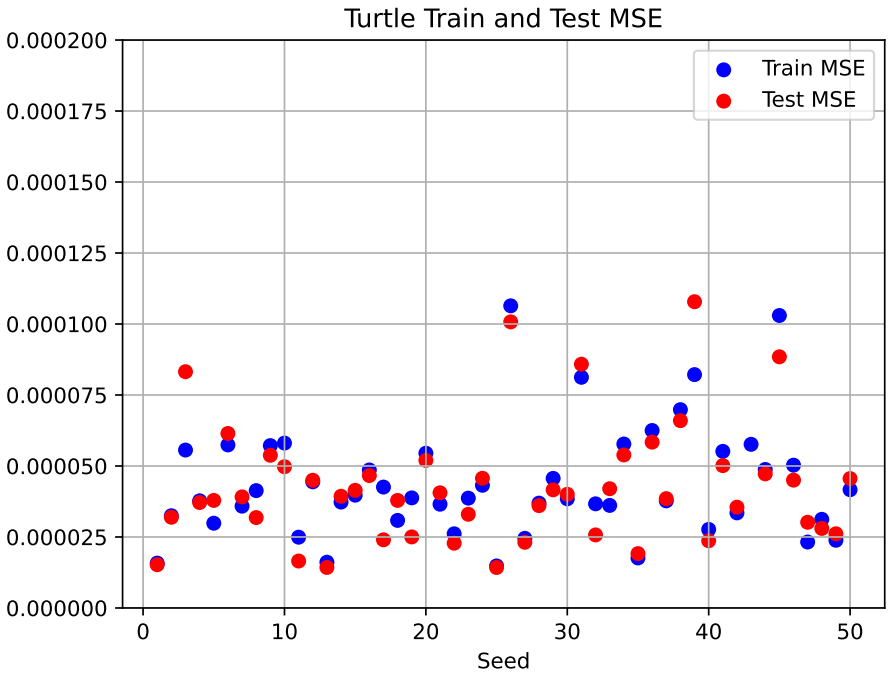

Figure 4.4 Turtle Bot Train and Test MSE

From the table, it is noticeable that there was one overfitted run and the test MSE is enormous because of that. If we look at the median, we can see that this was an exception. Otherwise, the results are acceptable.

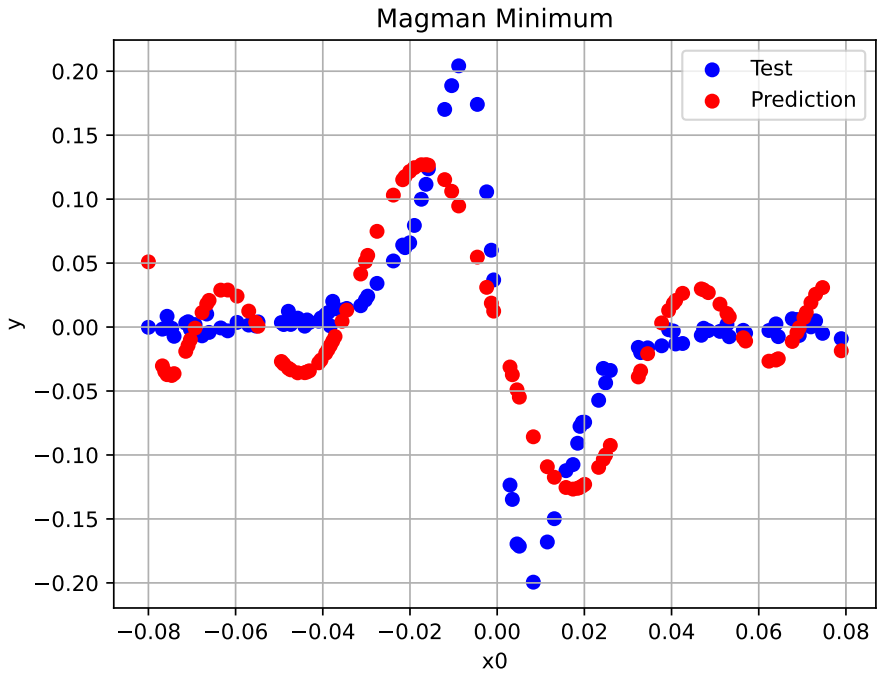

Figure 4.5 Magman with the minimum as aggregation function for node fitness

4.2 Evaluation Metric

Another experiment is the different evaluation metrics used for node fitness. The proposal was minimum, average and median.

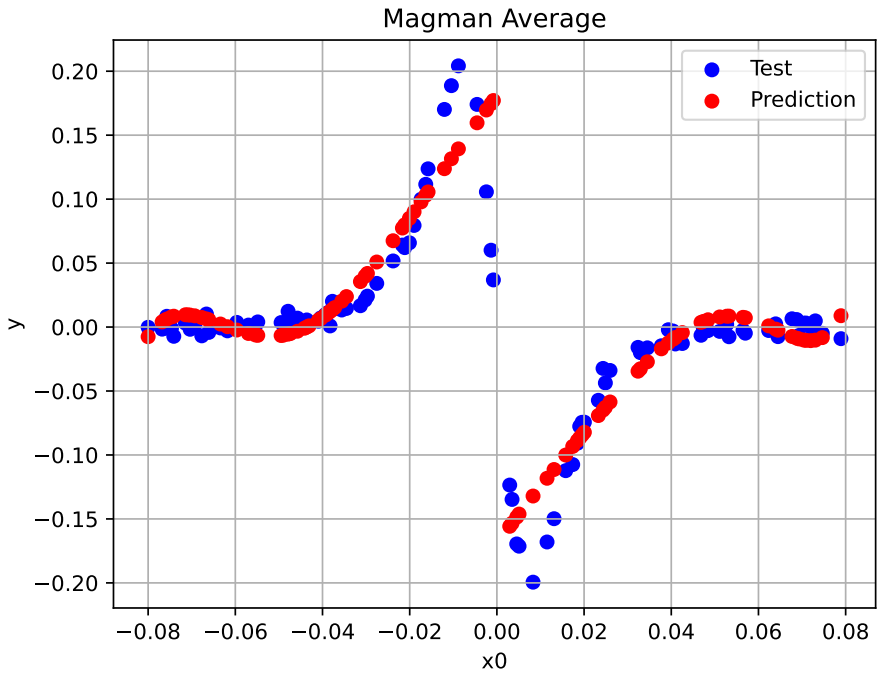

Figure 4.6 Magman with the average as aggregation function for node fitness

Taking a closer look, the median and minimum methods look very similar. On the other hand, the average took a different approach. Let's look closer at the tables
Judging from the tables, differences are negligible.

Table 4.4 MSE Magman Median Results

	Train MSE	Test MSE
Mean	0.001257	0.001035
Median	0.000853	0.000652
Standard Deviation	0.000762	0.000640
Minimum	0.000360	0.000345
Maximum	0.002304	0.001942

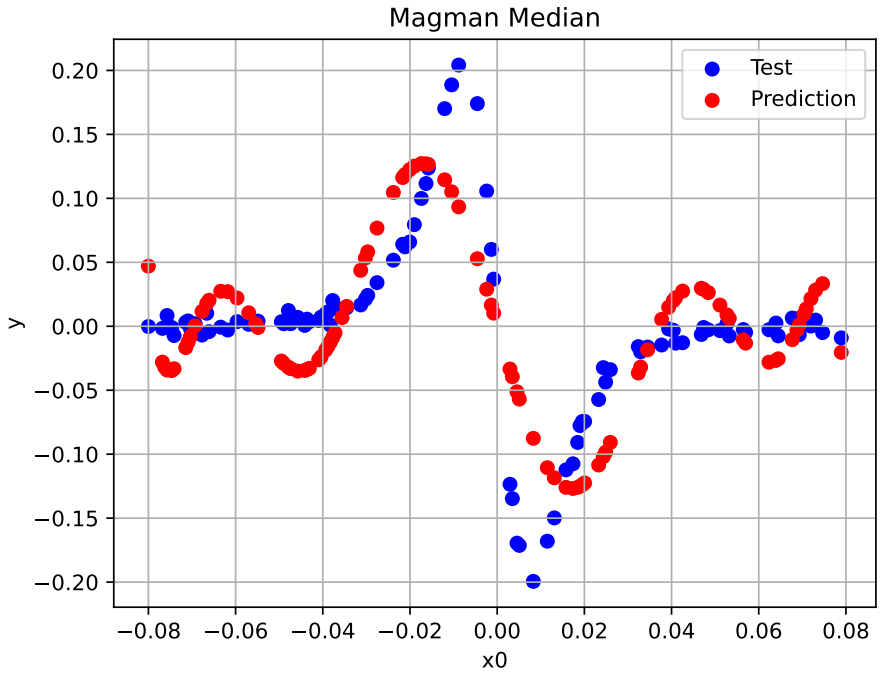

Figure 4.7 Magman with the median as aggregation function for node fitness

Table 4.5 MSE Magman Minimum Results

	Train MSE	Test MSE
Mean	0.001264	0.001055
Median	0.000975	0.000694
Standard Deviation	0.000774	0.000639
Minimum	0.000146	0.000148
Maximum	0.002313	0.001933

Table 4.6 MSE Magman Average Results

	Train MSE	Test MSE
Mean	0.001429	0.001202
Median	0.001463	0.001324
Standard Deviation	0.000760	0.000638
Minimum	0.000224	0.000209
Maximum	0.002305	0.002008

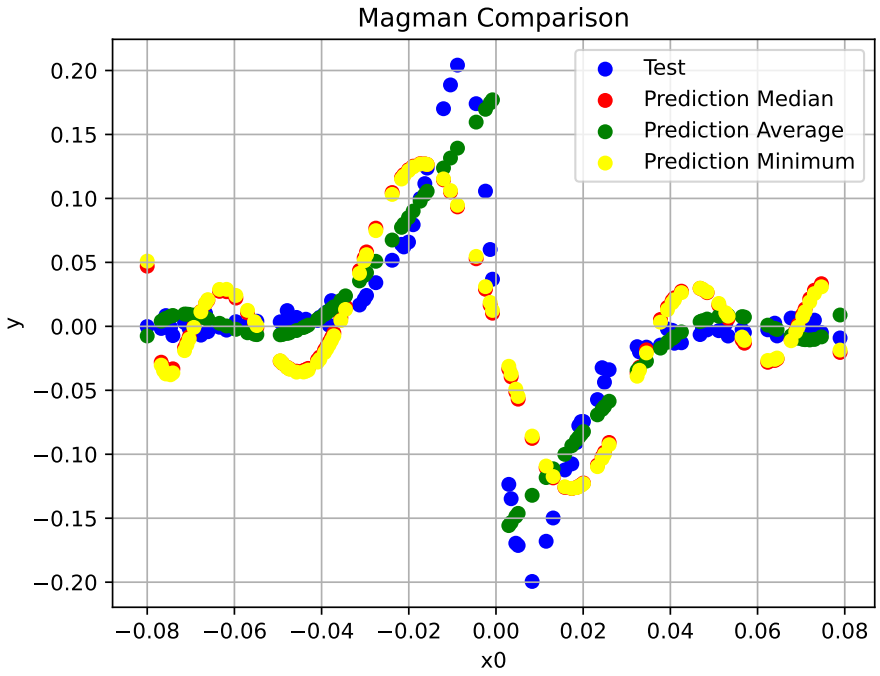

Figure 4.8 Magman Plots

Chapter 5
Conclusion

The proposed method works in given conditions; it is essential to realise that evolving a second population takes some time. Even though I did not measure performance differences, it took around twice the time as before. The conducted research was for single objective optimisation, which is not the main scope of the previous work. Different from some deep neural networks, conducted experiments yielded good results without the tedious fine-tuning of parameters, and models took a short time to produce. The correlation between individual features might be too complex to properly implement as the linkage[9] is hard to grasp. Judging from the last experiment, I do not deem this as a good direction to continue, as feature evolution can be highly unpredictable. The complexity of the models was of the correct size. As I could spot, any larger would lead only to very complex models, further away from the ideal formula. This can be said as my primary research focused on magnetic manipulation, which has only one variable, and resistors, which have elementary formulas.

5.1 Future work

The work could be extended to multi-objective optimisation, considering the prior knowledge and introducing constraints[4]. Longer training times, more extensive datasets and more complex models could be introduced to increase the likelihood of finding a model with good fitness. More complex models could tackle better problems like turtle bot or inverted pendulum[41], which are more complex. More methods to sustain diversity could be deployed, like diversity metrics or encouraging the models to choose nodes which were never used. The work could be compared to the N4SR[34] as I had the honour to test the previous multi-objective version of this SNGP and the N4SR.

Bibliography

1. AUGUSTO, Douglas Adriano; BARBOSA, Helio JC. Symbolic regression via genetic programming. In: *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*. IEEE, 2000, pp. 173–178.
2. WANG, Yiqun; WAGNER, Nicholas; RONDINELLI, James M. Symbolic regression in materials science. *MRS Communications*. 2019, vol. 9, no. 3, pp. 793–805.
3. KUBALÍK, Jiří; DERNER, Erik; BABUŠKA, Robert. Enhanced symbolic regression through local variable transformations. In: *Proceedings of the 9th International Joint Conference on Computational Intelligence*. 2017, vol. 1, pp. 91–100.
4. KUBALÍK, Jiří; DERNER, Erik; BABUŠKA, Robert. Symbolic regression driven by training data and prior knowledge. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 2020, pp. 958–966.
5. KUBALÍK, Jiří; DERNER, Erik; ŽEGKLITZ, Jan; BABUŠKA, Robert. Symbolic regression methods for reinforcement learning. *IEEE Access*. 2021, vol. 9, pp. 139697–139711.
6. BYRNE, Barbara M. *Structural equation modeling with EQS: Basic concepts, applications, and programming*. Routledge, 2013.
7. KOZA, John R; ANDRE, David; KEANE, Martin A; BENNETT III, Forrest H. *Genetic programming III: Darwinian invention and problem solving*. Vol. 3. Morgan Kaufmann, 1999.
8. SHARMA, s; CHAHAR, Vijay. A Comprehensive Review on Multi-objective Optimization Techniques: Past, Present and Future. *Archives of Computational Methods in Engineering*. 2022, vol. 29, p. 3. Available from DOI: 10.1007/s11831-022-09778-9.
9. LUKE, Sean. *Essentials of metaheuristics*. 2009.
10. LANTZ, Brett. *Machine learning with R: expert techniques for predictive modeling*. Packt publishing ltd, 2019.
11. Y., Chen; H., Jiang; C., Li; Z., Jiang; P., Ghamisi. Deep Feature Extraction and Classification Of Hyperspectral Images Based On Convolutional Neural Networks. *IEEE Trans. Geosci. Remote Sensing*. 2016, vol. 54, pp. 6232–6251. Available from DOI: 10.1109/tgrs.2016.2584107.
12. Y., Xu; R., Goodacre. On Splitting Training and Validation Set: A Comparative Study Of Cross-validation, Bootstrap And Systematic Sampling For Estimating The Generalization Performance Of Supervised Learning. *J. Anal. Test.* 2018, vol. 2, pp. 249–262. Available from DOI: 10.1007/s41664-018-0068-2.
13. A., Song L. Smola A. Gretton; K., Borgwardt; J., Bedo. Supervised Feature Selection Via Dependence Estimation. 2007. Available from DOI: 10.1145/1273496.1273600.

14. J., Suykens; J., Brabanter; L., Lukas; J., Vandewalle. Weighted Least Squares Support Vector Machines: Robustness and Sparse Approximation. *Neurocomputing*. 2002, vol. 48, pp. 85–105. Available from DOI: 10.1016/s0925-23120100644-0.
15. ŽEGKLITZ, Jan; POŠIK, Petr. Benchmarking state-of-the-art symbolic regression algorithms. *Genetic programming and evolvable machines*. 2021, vol. 22, pp. 5–33.
16. WEBB, G. Opus: An Efficient Admissible Algorithm For Unordered Search. *jair*. 1995, vol. 3, pp. 431–465. Available from DOI: 10.1613/jair.227.
17. ZHANG, R.; LENSEN, A.; SUN, Y. Speeding Up Genetic Programming Based Symbolic Regression Using Gpus. 2022, pp. 519–533. Available from DOI: 10.1007/978-3-031-20862-1_38.
18. CHEN, Q.; ZHANG, M.; XUE, B. Feature Selection To Improve Generalization Of Genetic Programming For High-dimensional Symbolic Regression. *IEEE Trans. Evol. Computat.* 2017, vol. 21, pp. 792–806. Available from DOI: 10.1109/tevc.2017.2683489.
19. MERTIKOPOULOS, P.; PAPADIMITRIOU, C.; PILIOURAS, G. Cycles In Adversarial Regularized Learning. 2018, pp. 2703–2717. Available from DOI: 10.1137/1.9781611975031.172.
20. VUL, E.; GOODMAN, N.; GRIFFITHS, T.; TENENBAUM, J. One and Done? Optimal Decisions From Very Few Samples. *Cogn Sci*. 2014, vol. 38, pp. 599–637. Available from DOI: 10.1111/cogs.12101.
21. PETERSEN, B.; LANDAJUELA, M.; MUNDHENK, T.; SANTIAGO, C.; KIM, S.; KIM, J. Deep Symbolic Regression: Recovering Mathematical Expressions From Data Via Risk-seeking Policy Gradients. 2019. Available from DOI: 10.48550/arxiv.1912.04871.
22. MITCHELL, M.; WU, S.; ZALDIVAR, A.; BARNES, P.; VASSERMAN, L.; HUTCHINSON B.and ... Gebru, T. Model Cards For Model Reporting. 2019. Available from DOI: 10.1145/3287560.3287596.
23. KOZA, J. Genetic Programming As a Means For Programming Computers By Natural Selection. *Stat Comput*. 1994, vol. 4. Available from DOI: 10.1007/bf00175355.
24. AZMI, E.; EHRET, U.; WEIJS, S.; RUDDELL, B.; PERDIGÃO, R. Technical Note: Bit By Bit: a Practical And General Approach For Evaluating Model Computational Complexity Vs. Model Performance. 2020. Available from DOI: 10.5194/hess-2020-128.
25. ZHAO, Q.; HASTIE, T. Causal Interpretations Of Black-box Models. *Journal of Business Economic Statistics*. 2019, vol. 39, pp. 272–281. Available from DOI: 10.1080/07350015.2019.1624293.
26. BUN, J.; BOUCHAUD, J.; POTTERS, M. Cleaning Large Correlation Matrices: Tools From Random Matrix Theory. *Physics Reports*. 2017, vol. 666, pp. 1–109. Available from DOI: 10.1016/j.physrep.2016.10.005.
27. WANG, Z.; ZHAN, Z.; LIN, Y.; YU, W.; YUAN, H.; GU, T.; ZHANG, J. Dual-strategy Differential Evolution With Affinity Propagation Clustering For Multimodal Optimization Problems. *IEEE Trans. Evol. Computat.* 2018, vol. 22, pp. 894–908. Available from DOI: 10.1109/tevc.2017.2769108.
28. LIU, Y.; GONG, D.; SUN, J.; JIN, Y. A Many-objective Evolutionary Algorithm Using a One-by-one Selection Strategy. *IEEE Trans. Cybern.* 2017, vol. 47, pp. 2689–2702. Available from DOI: 10.1109/tcyb.2016.2638902.
29. ALBADR, M.; TIUN, S.; AYOB, M.; AL-DHIEF, F. Genetic Algorithm Based On Natural Selection Theory For Optimization Problems. *Symmetry*. 2020, vol. 12, p. 1758. Available from DOI: 10.3390/sym12111758.

30. AIVALIOTIS-APOSTOLOPOULOS, P.; LOUKIDIS, D. Swarming Genetic Algorithm: a Nested Fully Coupled Hybrid Of Genetic Algorithm And Particle Swarm Optimization. *PLoS ONE*. 2022, vol. 17, e0275094. Available from DOI: 10.1371/journal.pone.0275094.

31. O'NEILL, Michael. *Riccardo Poli, William B. Langdon, Nicholas F. McPhee: A Field Guide to Genetic Programming: Lulu. com, 2008, 250 pp, ISBN 978-1-4092-0073-4*. Springer, 2009.

32. OSTERTAGOVÁ, Eva. Modelling using polynomial regression. *Procedia Engineering*. 2012, vol. 48, pp. 500–506.

33. SMOLA, Alex J; SCHÖLKOPF, Bernhard. A tutorial on support vector regression. *Statistics and computing*. 2004, vol. 14, pp. 199–222.

34. KUBALÍK, Jiří; DERNER, Erik; BABUŠKA, Robert. Toward Physically Plausible Data-Driven Models: A Novel Neural Network Approach to Symbolic Regression. *IEEE Access*. 2023.

35. CARLIN, Bradley P; LOUIS, Thomas A. *Bayesian methods for data analysis*. CRC press, 2008.

36. JACKSON, David. A new, node-focused model for genetic programming. In: *Genetic Programming: 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings 15*. Springer, 2012, pp. 49–60.

37. VAFAEE, Fatemeh; TURÁN, György; NELSON, Peter C; BERGER-WOLF, Tanya Y. Balancing the exploration and exploitation in an adaptive diversity guided genetic algorithm. In: *2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2014, pp. 2570–2577.

38. QI, Xiaofeng; PALMIERI, Francesco. Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space. Part II: Analysis of the diversification role of crossover. *IEEE Transactions on Neural Networks*. 1994, vol. 5, no. 1, pp. 120–129.

39. HURÁK, Zdeněk; ZEMÁNEK, Jiří. Feedback linearization approach to distributed feedback manipulation. In: *2012 American Control Conference (ACC)*. IEEE, 2012, pp. 991–996.

40. KUBALÍK, Jiří; DERNER, Erik; BABUŠKA, Robert. Multi-objective symbolic regression for physics-aware dynamic modeling. *Expert Systems with Applications*. 2021, vol. 182, p. 115210.

41. DERNER, Erik; KUBALÍK, Jiří; ANCONA, Nicola; BABUŠKA, Robert. Constructing parsimonious analytic models for dynamic systems via symbolic regression. *Applied Soft Computing*. 2020, vol. 94, p. 106432.