Development of Symbolic Models Using Genetic Programming and Domain Knowledge

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Julia Reuter, M.Sc.

geb. am 16.09.1996 in Bamberg

Gutachterinnen/Gutachter

Prof. Dr.-Ing. Sanaz Mostaghim
Prof. Dr. Bing Xue
Prof. Dr. Leonardo Vanneschi

Magdeburg, den 18.06.2025

Julia Reuter

Development of Symbolic Models Using Genetic Programming and Domain Knowledge

Dissertation, Otto von Guericke University
Magdeburg, 2025.

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 17.02.2025

Julia Reuter

Abstract

Problems from the science and engineering area come with challenging characteristics: They are often high-dimensional, highly non-linear, and have not yet been solved by humans due to their enormous complexity. While "black-box" machine learning methods such as deep neural networks can achieve high accuracies on such problems, the underlying relations remain opaque. However, it is essential for experts in science and engineering to analyze and understand the learned models. Symbolic regression (SR) is the construction of mathematical expressions from data to identify the relation between input and target variables. While traditional regression methods assume an underlying model structure, SR learns both the model structure and its associated parameters. Genetic programming (GP), a method from the family of evolutionary algorithms, is a widely used method for SR, because of its "white box" nature and its capability to optimize multiple objectives simultaneously. Often, domain knowledge is available that can contribute to discovering novel symbolic models. The goal of this thesis is to develop symbolic regression algorithms that can tackle complex science and engineering problems by making use of the valuable knowledge provided by domain experts.

In the related literature, various approaches to integrating domain knowledge into algorithms have been proposed and applied to real-world applications. This thesis aims to close a gap in this field by outlining and classifying these methods. Moreover, various techniques have been proposed to improve components of the GP algorithm that are relevant in practice, some of which are further improved upon in this thesis.

This thesis proposes two benchmark problems from robotics and fluid mechanics, and establishes a comparative baseline to evaluate the efficacy of the newly developed methods. To reduce the number of features of the high-dimensional problems, an inductive bias fitting the nature of the problem is proposed. Given the high complexity of the approached problems and the non-deterministic nature of the GP algorithm, methods to improve the repetition stability of a GP algorithm are additionally investigated. Furthermore, this thesis proposes methods to fulfil important expert requirements, such as methods to handle physical unit constraints. In this context, multi-objective optimization plays a pivotal role, allowing for the exploration of a diverse set of solutions while effectively optimizing multiple criteria. Empirical evaluations and case studies considering various problems with both known and unknown equations from the engineering and science area validate the proposed approaches. The results demonstrate how domain knowledge can improve the accuracy of symbolic models, while tackling increased problem complexities and developing meaningful equations for domain experts.

Zusammenfassung

Probleme aus den Bereichen der Wissenschaft und des Ingenieurswesens weisen anspruchsvolle Eigenschaften auf: Sie sind oft hochdimensional, in hohem Maße nichtlinear und wurden aufgrund ihrer enormen Komplexität bisher von Menschen nicht gelöst. Während „Black-Box"-Methoden des maschinellen Lernens wie tiefe neuronale Netze bei solchen Problemen hohe Genauigkeiten erreichen können, bleiben die zugrunde liegenden Beziehungen undurchsichtig. Für Experten in Wissenschaft und Technik ist es jedoch unerlässlich, die gelernten Modelle zu analysieren und zu verstehen. Symbolische Regression (SR) ist die Konstruktion mathematischer Ausdrücke auf Grundlage von Daten, um die Beziehung zwischen Eingabe- und Zielvariablen zu ermitteln. Während traditionelle Regressionsmethoden von einer zugrunde liegenden Modellstruktur ausgehen, lernt SR sowohl die Modellstruktur als auch die zugehörigen Parameter. Genetische Programmierung (GP), eine Methode aus der Familie der Evolutionären Algorithmen, ist aufgrund ihrer „White-Box"-Natur und der Fähigkeit, mehrere Zielfunktionen gleichzeitig zu optimieren, eine viel verwendete Technik für SR. Oft ist Domänenwissen vorhanden, das zur Entdeckung neuer symbolischer Modelle beitragen kann. Das Ziel dieser Arbeit ist es, symbolische Regressionsalgorithmen zu entwickeln, die komplexe technische und physikalische Probleme lösen können, indem sie das wertvolle Wissen von Domänenexperten nutzen.

In der einschlägigen Literatur wurden verschiedene Ansätze zur Integration von Domänenwissen in Algorithmen vorgeschlagen und auf reale Probleme angewendet. Diese Arbeit soll eine Lücke in diesem Bereich schließen, indem sie diese Methoden skizziert und klassifiziert. Darüber hinaus sind verschiedene Techniken zur Verbesserung praxisrelevanter Komponenten des GP-Algorithmus vorgeschlagen worden, von denen einige in dieser Arbeit weiter verbessert werden.

In dieser Arbeit werden zwei Benchmark-Probleme aus der Robotik und der Strömungsmechanik eingeführt und eine vergleichende Baseline erarbeitet, um die Wirksamkeit der neu entwickelten Methoden zu bewerten. Um die Anzahl der Variablen von hochdimensionalen Problemen zu reduzieren, wird ein induktiver Bias vorgeschlagen, der zu der Natur des Problems passt. Angesichts der hohen Komplexität der behandelten Probleme und der nicht deterministischen Natur des GP-Algorithmus werden darüber hinaus Möglichkeiten zur Verbesserung der Wiederholungsstabilität eines GP-Algorithmus untersucht. Zudem werden in dieser Arbeit Methoden vorgeschlagen, um wichtige Anforderungen von Experten zu erfüllen, wie z.B. Methoden zum Umgang mit Nebenbedingungen durch physikalische Einheiten. Dabei spielt die multikriterielle Optimierung eine zentrale Rolle, die es ermöglicht, eine Vielzahl von Lösungen zu untersuchen und gleichzeitig mehrere Zielfunktionen effektiv zu optimieren. Empirische Auswertungen und Fallstudien zu verschiedenen Problemen mit bekannten und unbekannten Gleichungen aus dem ingenieurtechnis-

chen und wissenschaftlichen Bereich validieren die vorgeschlagenen Ansätze. Die Ergebnisse zeigen, wie Domänenwissen die Genauigkeit symbolischer Modelle verbessern kann, während gleichzeitig eine höhere Problemkomplexität bewältigt und sinnvolle Gleichungen für Domänenexperten entwickelt werden.

Contents

Part I

Background

1 Introduction

1.1 Motivation

As the availability of datasets and computing power has increased over the last decades, so has the need to understand and describe the underlying relationships contained in such datasets. A central task in data analysis is regression: finding a model that predicts a response variable from input variables. This task is one of the most important in *machine learning* (ML) and has applications in a wide range of scientific disciplines, engineering, economics, finance, industry, and healthcare — in fact, in all disciplines that seek to describe reality using mathematical models. Many methods have been introduced in the past for regression tasks, such as simple linear regression, multiple linear regression, decision trees, Gaussian process regression, symbolic regression, as well as artificial neural networks and deep learning. Some methods assume a certain functional form *a priori*, which may be unsuitable for the problem at hand, or have limited interpretability.

Science and engineering disciplines have a long history of empirical discovery of equations with immense impact on today's research. Important fundamental laws such as Kepler's discovery of planetary orbits around the sun or the Newtonian law of gravitation were developed empirically through iterative measurements and trial and error on potential functions that describe the observed behavior. Problems from the science and engineering area come with challenging characteristics: they are often high-dimensional, have highly nonlinear underlying behavior, and sometimes only noisy measurements or data are available. Many problems have not been solved yet by humans due to their enormous complexity. However, domain knowledge about the underlying problem as well as the fundamentals of the discipline are often available, which can be beneficial when solving more complicated problems.

When approaching such complex problems with regression algorithms, it is essential for domain experts to analyze and understand the learned models, and to avoid unexpected behaviors. To this end, combining ML techniques with domain knowledge is a promising approach. For "black-box" methods such as deep learning, various techniques have been proposed to enforce desired behaviors in the models [126]. While they can achieve high accuracies for such problems, the underlying relationships remain opaque. Since it is impossible to test the entire possible input space of variables to such a system, some level of uncertainty in the behavior will always remain. *Symbolic regression* (SR), on the other hand, is a method for modeling complex relationships between variables by discovering the underlying mathematical expressions. SR algorithms are particularly useful when the functional relationship between variables is unknown or cannot be easily modelled using traditional regression methods. Free-form

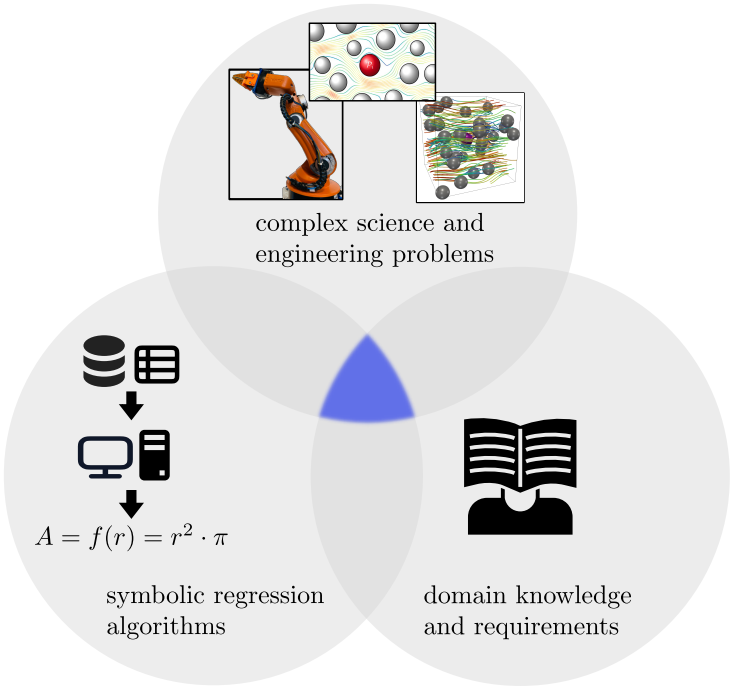


Figure 1.1: The big picture of this thesis. The contributions of this thesis are located in the blue intersected area.

equations are generated from data, which allow domain experts to analyze the behavior of the underlying systems. The advantages of symbolic models over traditional and "black box" regression models are obvious, but the path to developing a symbolic model can be cumbersome in practice. While the accuracy of such models is obviously an important measure of quality, other characteristics are also of practical importance. These include whether the expression follows physical laws, whether it resembles the expected physical behavior, and whether it contains components that have been used in expressions for similar problems. In other words, symbolic models must not only be accurate, but also satisfy certain constraints and be appropriate for the problem at hand. Several criteria need to be met to achieve a useful result.

Genetic programming (GP) from the family of *evolutionary algorithms* (EAs) is an established method for SR. Guiding the search with a heuristic method is advantageous given the potentially huge search space of variable-length equations. GP allows for *multi-objective optimization* (MOO) to optimize multiple criteria simultaneously while searching for equations. Rather than presenting a single solution, this method allows evolving a set of optimal and near-optimal solutions for domain experts to choose from.

The focus of this thesis is to bring together the three components introduced in the previous paragraphs: the inclusion of domain knowledge in GP algorithms to solve complex problems from science and engineering, which have a strong requirement for symbolic models. Fig. 1.1 shows the intersection of research areas in which the contributions of this thesis are located. Out-of-the-box GP algorithms often do not provide satisfactory results in terms of accuracy or interpretability, which can lead to reduced acceptance of such models.

To overcome this issue, this thesis introduces and classifies different levels of domain knowledge, intending to improve the quality and usefulness of the resulting models. For many scientific and engineering applications, such equations are only trustworthy and useful if they reflect certain physical properties. In the further course of this thesis, algorithmic methods to supply domain knowledge at different levels to the GP algorithm are presented. Being unresolved issues in practical applications, the focus is on inductive bias to break down high-dimensional problems into smaller problems instances, as well as constraint handling techniques for problem-dependent constraints. For trustworthy models, a high success rate over several repeated runs is furthermore

desirable. The methods are showcased with unsolved problems from robotics and fluid mechanics, but also consider other benchmark problems from thermodynamics and datasets from known physics equations that have been discovered empirically in the past.

1.2 Research Questions and Goals

The main goal of this thesis is to develop a class of learning algorithms based on data from experiments on highly complex problems with unknown ground truth, with the specific goal of integrating domain knowledge. The main purpose of the integration of domain knowledge is to produce physically meaningful equations. To achieve this goal, we apply algorithms to two problems from the area of fluid mechanics and robotics. Both exhibit high-dimensional properties, are currently not fully understood and resolved by humans, and provide reliable data from simulations or experiments. These can be modelled as graphs, and domain knowledge is available from experts in the fields. The availability and quality of data are not a major concern of this thesis, as high-fidelity datasets are available for both problems.

While these problems differ in their specific application area, they share certain characteristics. To significantly contribute to this field, it is important to develop algorithms that are flexible enough for diverse applications, yet adaptable to problem-specific properties. Thus, a central goal of this thesis is to develop methods to advance algorithms for SR problems in several aspects, which generalize well across different problem domains, but also offer adaptability to specific characteristics of certain problems. In this dissertation, we aim to illustrate the algorithmic advancements on both problems. To this end, this thesis addresses the following five research goals:

G 1: Develop algorithms that enhance the applicability and expressiveness of GP for symbolic regression for graph-representable problems involving physical measurements

G 2: Design benchmark problems from fluid mechanics and robotics with unknown ground truth and varying complexities

G 3: Establish a comparative baseline on the defined benchmarks by applying state-of-the-art GP algorithms to set a performance standard for advancements presented later in this thesis

G 4: Assess methods to improve the repetition stability of a GP algorithm

G 5: Evaluate the effectiveness of the proposed enhancements and compare them to the baseline

These research goals also outline the broad structure of this thesis. They resulted in seven research questions, which emerged from the challenges faced when applying state-of-the-art GP methods to identify meaningful and useful equations with unknown ground truth but with available domain knowledge. The first question concerns recent developments in the area of physics-informed machine learning. Additionally , we will examine GP, focusing on challenges when applied to real-world problems, as well as the integration of domain knowledge for science and engineering problems.

RQ 1: Which techniques exist to develop symbolic models for problems from science and engineering?

RQ 1.1: What are the current developments in the area of physics-informed machine learning?

RQ 1.2: What are the current challenges in GP for application to symbolic regression problems, and how are they addressed?

RQ 1.3: How is domain knowledge integrated into state-of-the-art GP techniques, and how can these techniques be classified?

This thesis studies the effectiveness of the proposed methods using two problems from real-world applications, one of which concerns the inverse kinematics of robotic manipulators. While reliable methods exist for standard manipulators, non-standard manipulators require more flexible approaches due to their unique configurations and constraints, which often render traditional analytical solutions impractical or inaccurate. We address the following research questions, aiming to identify symbolic models for the inverse kinematics of arbitrary manipulators, with a focus on suitable objective functions and the integration of domain knowledge:

RQ 2: How can symbolic models for the inverse kinematics of arbitrary robotic manipulators be developed?

RQ 2.1: What are suitable objective functions to achieve physically meaningful equations?

RQ 2.2: How do different types of domain knowledge integrated into the algorithm influence its performance?

RQ 2.3: What are the limitations of this approach?

Another problem used as a case study in this thesis is from the area of fluid mechanics, specifically the simulation of particle-laden flows, i.e., particles suspended in a fluid flow. Predicting the velocity field and forces acting on a particle in a flow involving more than one spherical particle remains an unresolved challenge in the literature. In the following research question, we assess the performance of a GP algorithm with integrated building blocks as a type of domain knowledge, and compare it against other methods from the literature.

RQ 3: How can symbolic models describing particle-laden flows be developed?

RQ 3.1: How can a GP algorithm benefit from building blocks provided by domain experts?

RQ 3.2: How do the evolved symbolic models perform against state-of-the-art baseline methods?

RQ 3.3: What are the limitations of this approach?

In practical applications, trustworthy algorithms are an important factor for the acceptance of the developed solutions. For GP to be considered a trustworthy tool, it is desirable for the algorithm to consistently deliver high-quality results across repeated runs. However, as evolutionary algorithms are inherently stochastic, achieving deterministic outputs is unrealistic. Their search process resembles a guided random walk through a vast search space, where results can vary significantly between executions. Despite this inherent randomness, it is possible to enhance the success rate of a GP algorithm, i.e., the likelihood that it consistently recovers meaningful correlations or solutions across multiple runs. Island models present a promising approach to this end. We address the interplay between island models and multi-objective optimization in the following research question:

RQ 4: Can the success rate of a GP algorithm be improved with island models?

RQ 4.1: How does the interplay between island models and multi-objective optimization influence the final results?

High-dimensional problems with large numbers of features present significant challenges for GP, primarily due to the exponential growth of the search space with an increasing number of features. While some studies define "large" as more than 50 features [e.g., 296], even smaller sets with around 15 or more features can significantly increase the difficulty for the algorithm to identify meaningful correlations. Both case studies addressed in this thesis deal with such problems, rendering them a challenging task for GP. However, both of these problems can be represented as graph structures, which enables addressing them with *graph neural network*s (GNNs). Recent advances in GNNs as an inductive bias for GP provide a promising method to tackle this issue by splitting the problems into smaller subproblems with smaller search spaces each [e.g. 60, 162]. Based on this, our research aims to explore the potential of GNNs as an inductive bias within GP for high-dimensional, graph-representable problems, focusing on the following questions:

RQ 5: What is the potential of graph neural networks as an inductive bias for GP to discover unknown symbolic models for high-dimensional problems that can be modeled as graphs?

- RQ 5.1: Can particle-laden flows be approximated with graph neural networks?
- RQ 5.2: How do the resulting equations perform in terms of error and interpretability?
- RQ 5.3: Can the inverse kinematics problem be learned with graph neural networks?

In science and engineering, problems are typically characterized by measurements with well-defined units, such as length in meters or time in seconds. To develop reliable symbolic models for these domains, GP must not only produce accurate results but also maintain unit consistency within its solutions. Accurate equations for these problems typically require both the learning and fitting of new constants and accordance with physical laws. Dimensional analysis enforces unit consistency in GP, helping to filter out nonphysical solutions and ensuring interpretability by aligning each term with real-world measures. New constants, however, have undetermined units, which complicates the dimensional analysis. The inclusion of free constants in dimensional analysis is addressed in the following research questions, together with methods to handle unit constraints within the algorithm.

RQ 6: How can unit-aware equations be developed that include free constants?

- RQ 6.1: How can undetermined units of free constants be considered during dimensional analysis?
- RQ 6.2: What are suitable methods to handle unit constraints?

One major topic of this thesis is to develop algorithms that can be tailored to specific problems and are also applicable to a wide range of application areas with similar characteristics. Using the two case studies from fluid mechanics and robotics, the last research question aims to assess the generalizability and adaptability of our algorithms. A comparison with the baseline methods established in RQ2 and RQ3, as well as identifying the limitations of these methods, are the focus here.

RQ 7: Are the developed algorithms flexible to be applicable to a wide range of problems and capable of being tailored to domain-specific characteristics simultaneously?

- RQ 7.1: How do the algorithmic advances presented in this thesis perform in predicting particle-laden flows and inverse kinematics, compared to the baseline methods?

RQ 7.2: What are the limitations of these methods?

The broader implications of this thesis are to close existing practical application gaps, and to improve GP for SR by addressing issues that are shortcomings in the current frameworks and literature, but of high relevance in practice. The ultimate goal is to develop models that are genuinely useful for domain experts in situations where there is no established ground truth.

1.3 Structure of this Thesis

In the following, the structure of this thesis will be described. It is divided into three parts, with the first part covering the background of this thesis, including the motivation and research goals already discussed in this chapter, as well as the fundamentals of ML and SR, with a special emphasis on GP, in Chapter 2. Subsequently, Chapter 3 addresses RQ1 and covers research relevant to the context of this thesis, specifically the application of ML algorithms to problems from science and engineering, recent advances in the area, as well as methods to integrate domain knowledge in GP algorithms.

In the second part, the two problems studied in this thesis from robotics and fluid mechanics are introduced, and algorithmic baselines are established. Chapter 4 addresses RQ2 and introduces the inverse kinematics problem for arbitrary robotic manipulators. This chapter places great emphasis on the selection of objective functions and the integration of domain knowledge, which are of crucial importance in the context of developing useful symbolic models. Chapter 5 discusses RQ3 and outlines the problem related to the simulation of particle-laden flows. We examine how the use of expert-defined building blocks influences the final result and compare against two widely accepted baseline methods from fluid mechanics and machine learning.

In the third part of this thesis, the proposed algorithmic advances are presented, targeting several aspects to improve GP algorithms for practical applications. Chapter 6 addresses RQ4 and studies the interplay between island model GP and multi-objective optimization, aiming at improving success rates of the algorithm across several repeated runs. In Chapter 7, we discuss RQ5 and explore the potential of GNNs as an inductive bias for GP, assessing their applicability to problems with shared characteristics from different application areas. New methods for unit-aware GP with free constants are introduced in Chapter 8, addressing RQ6 and completing the machine learning pipeline proposed in this thesis. We will apply the unit-aware GP approach to develop symbolic models for both benchmark problems. Although partially touched on in the previous chapters, Chapter 8 also addresses RQ7 and discusses the effectiveness of the proposed machine learning pipeline for the problems arising from robotics and fluid mechanics compared to the baseline methods introduced earlier.

2 Scientific Fundamentals

Chapter Structure

This chapter introduces the basic scientific concepts that are essential for the topics covered in this thesis. After an overview of machine learning methods and tasks, traditional regression methods are discussed. Furthermore, this chapter covers artificial neural networks, with graph neural networks as a category that is essential for the approaches presented in the course of this thesis. The basics of evolutionary algorithms are then introduced, with a focus on multi-objective optimization. Finally, the main concepts of genetic programming are introduced, which is the underlying method for the symbolic regression algorithms presented in this thesis.

2.1 Machine Learning

2.1.1 Machine Learning Methods and Tasks

Machine learning (ML) is an important part of the area of artificial intelligence [237]. As the amount of data available has increased drastically over the last decades, so has the requirement to process this data and generate knowledge from it [6]. Following the widely used terminology of Mitchell, ML can be defined as follows:

Machine Learning Definition

"A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [180]

A well-designed learning problem thus requires three components: a specific task T, a performance measure P, and a type of training experience E. Consequently, ML can be categorized into different types of learning, which are each suitable for specific learning tasks. Classified by the type of learning feedback, the most common ML types are supervised learning, unsupervised learning, and reinforcement learning.

Supervised Machine Learning

In supervised ML, the task T is to learn a mapping function f from inputs $\mathbf{x} \in \mathcal{X}$ to outputs $\mathbf{y} \in \mathcal{Y}$. The input is usually referred to as a feature, covariate or predictor [190]. It typically constitutes a real-valued vector of dimensionality or number of features D, so that $\mathcal{X} = \mathbb{R}^D$ The output, also called label or target, is known. The experience E is defined as pairs of inputs and the respective outputs, where the number of pairs N is called the sample size of a dataset.

Classification and regression are both supervised learning tasks. Classification maps the input space to an output space consisting of unordered and mutually exclusive labels, so that $\mathcal{Y} = \{1, 2, \ldots, C\}$. The task is to predict a discrete class label. Regression is the task of predicting a continuous, real-valued target quantity $\mathcal{Y} = \mathbb{R}$. The type of the output feature determines which performance

measures P are suitable for the problem. The overall goal is to develop a general model from a limited amount of data, which can be used to make predictions in the future [181].

Unsupervised Machine Learning

Unlike supervised learning, unsupervised learning methods are used when the output is not known in advance. The goal or learning task T is to "make sense of the data", i.e., identify underlying patterns, hidden structures or rules. The experience E typically refers to the raw input data without any labels or target variables [190]. Such techniques are useful for exploratory data analysis of large datasets. Density estimation or clustering and association rules are common tasks approached with unsupervised learning. Furthermore, dimension reduction techniques are applied to high-dimensional input spaces to reduce the dimensionality of the problem and capture the most essential characteristics of the data. Evaluating the result of unsupervised learning using a performance measure P is a complex task, as no ground truth is known [41].

Reinforcement Learning

Learning through interactions with the environment is the core idea behind *reinforceement learning* (RL) [265]. An agent is tasked with learning how to respond to an environmental state, or input $\mathbf{x}$, by taking an action, which is encoded through the policy $\mathbf{a} = \pi(\mathbf{x})$ [190]. The learning experience E includes the actions that the agent takes in the environment, the feedback it receives for this action, and the policy update based on the feedback. The feedback can be a reward or a penalty, depending on the positive or negative impact of the action performed [6].

RL is suitable for a broad range of applications, such as game playing, navigation and controlling tasks, as well as chatbots for conversations. However, as Murphy points out, "it can be harder to make RL work" [190, Chapter 1.4], with the main problem being that feedback is only given at specific time steps or once a goal is reached. This makes it difficult for the algorithm to identify which of the previous actions was the one that led to success. Overall, the trial-and-error search and delayed rewards are the most distinguishing characteristics of RL [265].

There exist multiple variations of these three ML types, including active learning, semi-supervised learning and transductive inference. The interested reader may refer to [181, 190] for additional information.

2.1.2 Classical Regression Methods

Regression Definition

Regression analysis is a fundamental supervised learning technique in ML. It involves modeling the relationship between one or more independent variables and a dependent variable. The goal is to understand and predict the behavior of the dependent variable [120].

Given one or multiple independent variables $\mathbf{x}$, the goal is to predict the response variable y as a function of $\mathbf{x}$ and the learned coefficients $\mathbf{w}$, so that

$$y = f(\mathbf{x}, \mathbf{w}) + \epsilon. \tag{2.1}$$

The error term ϵ in Eq. 2.1 denotes the variations in the data that cannot be predicted by the model. It indicates the existence of hidden variables that cannot be observed [6]. Regression techniques can be classified into linear and non-linear methods.

In addition to the data, the components required for the design of a regression algorithm, or an ML algorithm in general, include a model family, a cost function and an optimization algorithm [98, Chapter 5]. Often, the model family and learning task already imply the use of specific cost functions and appropriate optimization algorithms for the problem at hand. Further considerations on the design of ML algorithm are presented in Sec. 2.1.4. In the following, some classical regression methods are introduced. They can be broadly classified into parametric and non-parametric models. Parametric models generally

contain a fixed and finite number of parameters, regardless of the dataset size. Non-parametric models make fewer assumptions about the underlying distributions and can adapt the number of parameters depending on the problem and dataset size.

Linear Parametric Regression

The main characteristic of linear regression as used in this thesis is the assumption of a linear relationship *with respect to the parameters* $\mathbf{w}$.

Simple Linear Regression

Simple linear regression involves a linear combination of input variables

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} = w_0 + w_1x_1 + \cdots + w_Dx_D \tag{2.2}$$

with the parameter vector $\mathbf{w} = (w_0, w_1, \ldots, w_D)$, including the intercept w_0, and the feature vector $\mathbf{x} = (1, x_1, \ldots, x_D)$ [22]. The simple linear regression model is both linear regarding the parameters and the input variables. As a strict linear relationship between input and output variables poses limitations on the learned function, various adaptations of the simple linear model can be made.

Polynomial Regression

Polynomial regression functions are still linear with regard to the parameters, but allow for nonlinear transformation of the input variables. The relationship is modeled as a P^{th} degree polynomial function of an independent variable x [120].

$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + w_3x^3 + \cdots + w_Px^P \tag{2.3}$$

With increasing degree P, the number of parameters increases accordingly. In the general case, arbitrary nonlinear functions ϕ, also referred to as basis functions, can be applied to multiple input features $\mathbf{x}$, resulting in

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\phi(\mathbf{x}) = \sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x}) \tag{2.4}$$

where $\phi = (1, \phi_1, \ldots, \phi_{M-1})$ and $\mathbf{w} = (w_0, \ldots, w_{M-1})$ in the general case, with the total number of parameters denoted by M [22]. In the context of polynomial regression, the basis functions correspond to $\phi(x) = (1, x, x^2, \ldots, x^P)$.

Ordinary Least Squares

The goal of a regression algorithm is to tune the parameter values in a way that the distance between the function and the data samples is minimized [120]. The most widely used cost function to fit the parameters in a parametric linear regression model is *ordinary least squares* (OLS). OLS minimizes the *sum of squared residuals* (RSS), where the so-called residual ε_i is the squared difference between the response variable y_i and the prediction $\hat{y}_i$ made by the linear model on the input variables $\mathbf{x_i}$ on the sample i within the dataset [190].

$$RSS = \sum_{i=1}^{N} \varepsilon_i^2 = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T\mathbf{x}_i)\right)^2 \tag{2.5}$$

The feature vector $\mathbf{x}$ in Eq. 2.5 can either represent the original features as well as features that are transformed by a basis function ϕ. Solving an OLS is usually performed by setting the gradient of the RSS to zero and solving the system of equations. Typically, OLS for linear models can be solved analytically as it involves minimizing a convex, quadratic objective function with a closed-form solution [22]. The OLS problem can be formulated in matrix notation and solved with the help of linear algebra and the so-called normal equation. More details on the used calculus and matrix transformation can be found in [190, Chapter 11.2.2] and [120, Chapter 3].

Nonlinear Parametric Regression

Advanced Regression Techniques

To overcome the limited representation capacity of linear models, nonlinear models can be used. These are nonlinear with respect to the parameters, i.e., go beyond linear combinations of terms [229]. Popular examples include the exponential growth and decay model in Eq. 2.6, or the harmonic oscillator in Eq. 2.7.

$$y(x, \mathbf{w}) = w_0 \exp(w_1 x) \tag{2.6}$$

$$y(\theta, \mathbf{w}) = w_0 \sin(w_1 \theta + w_2) \tag{2.7}$$

Further examples of parametric nonlinear models that are commonly used in practice include the Michaelis-Menten kinetics, the logistic growth model, and the van der Waals equation. In many scientific areas, ranging from physics and engineering to economics and psychology, domain experts design custom nonlinear functions to tailor a model to the characteristics of a dataset. This allows to integrate domain knowledge, and model complex behaviors beyond the classical nonlinear parametric models [227].

Nonlinear Parameter Fitting

Once the shape of a model and the position of its coefficients are determined, the values of the coefficients are tuned using nonlinear regression algorithms. Next to the previously introduced RSS, two common cost functions for regression are the *mean-squared error* (MSE) and the *mean absolute error* (MAE).

$$MSE = \frac{1}{N} \sum_{1}^{N} (y_i - \hat{y}_i)^2 \tag{2.8}$$

$$MAE = \frac{1}{N} \sum_{1}^{N} |y_i - \hat{y}_i| \tag{2.9}$$

In the general case of arbitrary nonlinear functions, no analytical solution is guaranteed for the parameter estimation. To overcome this issue, iterative numerical optimization algorithms are often employed to identify model parameters that minimize the cost function. Prominent algorithms are *gradient descent* (GD), the Newton-Rhapson method as well as the Levenberg-Marquardt algorithm, which is specifically tailored to solve nonlinear least squares problems. Due to the nonlinear relations, two major challenges in fitting parameters in a nonlinear model are the choice of the starting values of the parameter, as well as convergence to a global rather than a local optimum. These are often approached by sophisticated parameter initialization procedures and repeated execution of the algorithm from different starting conditions. Other methods for parameter estimation include *maximum likelihood estimation* (MLE) and *Bayesian inference*, which also often involve numerical techniques, as well as heuristic optimization algorithms like *particle swarm optimization* (PSO) and *genetic algorithms* (GA). More detailed information on iterative algorithms for nonlinear least squares can be found in [23, 193, 227].

Regularization Techniques

Frequently used extensions of the least squares method to address the disadvantages of having many terms and coefficients in the regression model are ridge regression and lasso regression. Ridge regression tunes the coefficients β so that the RSS (or any other cost function) with an added penalty term is minimized, which reads as follows:

$$min \left(RSS + \lambda \sum_{j=1}^{p} \beta_j^2 \right) \tag{2.10}$$

The penalty term becomes less severe the closer β_i gets to zero. The parameter $\lambda > 0$ is called the shrinkage parameter and is determined beforehand [120]. It regularizes the impact of the penalty in relation to the RSS. Depending on the choice of λ, different estimates for the coefficients β are produced. Overall, this shrinkage method is used to obtain smaller, but non-zero coefficient values, which is expected to reduce the model variance, especially as the number of predictors approaches the number of data samples.

Lasso regression uses a similar penalty term as ridge regression.

$$min \left(RSS + \lambda \sum_{j=1}^{p} |\beta_i| \right) \tag{2.11}$$

The absolute value of the coefficients in the penalty term forces some coefficients to be equal to zero. In this way, the overall model size can be reduced by eliminating all terms which are multiplied by zero. This results in more spare models compared to the ones generated with ridge regression [120].

Multilayer Perceptron Regression

A *multilayer perceptron* (MLP) is a special case of parametric nonlinear model and has become a powerful and predominant regression method for complex nonlinear problems. MLPs are inspired by the biology of the human brain. The smallest unit of an MLP is a single perceptron, which is connected to other perceptrons by links. Each perceptron aggregates its inputs and applies a usually nonlinear activation function to it. The hierarchical structure with multiple layers (depth) of a certain number of perceptrons (width) each and the links between them results in a complex *artificial neural network* (ANN) architecture.

Parametric regression models as introduced at the beginning of this subsection can be considered "white-box" models, i.e., they are based on mathematical equations that are transparent regarding the nonlinear transformation applied to the input variables. ANNs, on the other hand, are often classified as "black-box" models, as they involve complex architectures and transformations that are more difficult to interpret. However, these characteristics enable ANNs to perform particularly good at capturing complex patterns and nonlinearities within data. Sec. 2.2 provides more detailed insights into the principles of ANNs.

Non-parametric Regression

Other than the previously introduced parametric models, non-parametric models are more flexible in their functional form, and the number of parameters can grow with the size of the dataset. Examples for non-parametric nonlinear models include Kernel methods [251] such as support vector regression [277], as well as Gaussian process regression [217].

Support Vector Regression

Support vector machines (SVM) were originally proposed for classification problems, with the idea of separating two classes by maximizing the margin between a hyperplane and the closest data point. The extension to *support vector regression* (SVR) was proposed in [277], intending to minimize the prediction error between the data samples and the computed hyperplane within a predefined margin. An SVR model takes on the form

$$Y = \beta_0 + \sum_{n:\alpha_n>0} \alpha_n \mathcal{K}(\mathbf{x_n}, \mathbf{x}) \tag{2.12}$$

where $\mathcal{K}(\mathbf{x_n}, \mathbf{x})$ is the so-called Kernel function [190]. A Kernel function maps data points into a higher dimensional space, intending to capture complex patterns between the input and response variables. Kernel functions can apply linear as well as nonlinear transformation, as a collection of popular regression

Kernels in [277, 251, 184] shows. In this thesis, we are mainly concerned with the design of algorithms to develop nonlinear parametric models. However, as also the influence of domain knowledge is an important topic in this thesis, the example of SVR for kernel-based methods demonstrates that expert knowledge is essential across various applications.

This section introduced the fundamental and most relevant regression methods for the further course of this thesis. However, the area of regression methods is huge, with each method featuring special characteristics and application areas. For a more profound understanding of the presented as well as further methods, the interested reader may refer to [120, 181, 190].

2.1.3 Symbolic Regression

The Core Idea

The choice of a model type in classical regression has an enormous impact on its performance. For decades, and still nowadays, many equations in science, engineering, physics, economics and various other disciplines are developed by hand from observations about systems. This process typically involves several iterations of adapting an initial guess model to better fit the data. Famous fundamental physics principles such as Kepler's discovery of planetary orbits around the sun or the Newtonian law of gravitation were developed empirically through iterative measurements and trial and error on potential functions that describe the observed behavior. *Symbolic regression* (SR), on the other hand, makes learning a proper model structure part of the learning task. Other than classical regression methods, no functional form is assumed beforehand, which makes the algorithms very flexible in terms of application areas. At the same time, the search is slowed down by the need to fit not only the parameters as in classical regression, but also to vary over the model structure, which theoretically spans an infinite number of possible model structures.

History and Development

The first programs for automated symbolic regression were developed in 1977 with the introduction of a tool named BACON [159]. The use of genetic algorithms and specifically genetic programming for symbolic regression was first proposed by Koza in 1992 [140]. Evolutionary methods are probably the most intensively studied algorithms for symbolic regression, and therefore also the main focus of this thesis. Sec. 2.4 gives an overview of the fundamentals of genetic programming algorithms for symbolic regression. More recent developments in this area are covered in Sec. 3.2.4 and 3.3.

Symbolic Model Types

A classification of mathematical model types relevant to science and engineering can be found in [227, Chapter 2]. This thesis contributes to the development of algorithms that identify symbolic models that are on the spectrum between *empirical* and *phenomenological* models. While empirical equations are purely data-driven and used to make predictions in the future, phenomenological models usually incorporate theoretical insights in the modeling process and aim to provide a more profound understanding of the underlying patterns [227]. When domain knowledge *and* data is available, the developed models lie between these two extremes.

2.1.4 Design Decisions for Regression

This thesis is concerned with regression problems and models. Therefore, this section discusses high-level design decisions that must be considered to approach a regression problem.

Choice of Model

A regression model maps a set of input features to a real-valued output space $\mathbf{y} \in \mathbb{R}$. Choosing an appropriate underlying model to solve a regression problem is a non-trivial task, as the big variety of available models, some of which were introduced previously, indicates. Given a hypothesis class $\mathcal{H}$, a hypothesis, or function $h \in \mathcal{H}$ is chosen by a learning algorithm using the training set [6]. The hypothesis class $\mathcal{H}$ determines the underlying model [181], for example a linear model or a polynomial model of a specific degree. The decision about a hypothesis class is made at the design step of the ML system. Each hypothesis

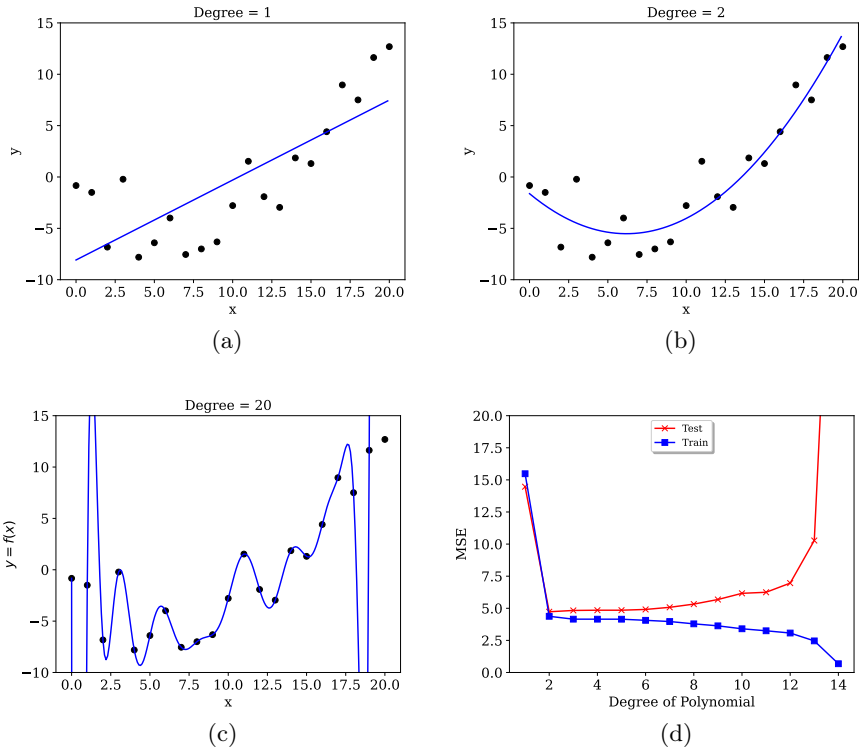


Figure 2.1: Polynomials of different degrees fitted through data points. The lower right subplot depicts the MSE values of training and test datasets over increasing polynomial degrees. Code to generate the plots inspired by the codebase [191] related to [190].

can be approximated by various learning algorithms which tune the hypothesis parameters. Every learning algorithm comprises hyperparameters which are determined beforehand.

According to the "no free lunch" theorem, there is no single best model that performs best across all problem domains [294]. Many models assume a certain functional form beforehand, such as linear regression, which will always produce a linear output. This so-called inductive bias is a prior assumption about the model type that is expected to work well on a specific problem. If this assumption proves incorrect, the performance may suffer. Even within the same model family, different configurations may perform better or worse, depending on the choice of learning algorithm and hyperparameters [181]. Fig. 2.1(a)-(c) shows how polynomials of different degrees are fitted through a set of data points. It becomes apparent that the degree of the polynomial determines how well the curve fits the data and how much the curve adapts to the variations in the data points.

Next to trial and error, a fitting model type can be identified by making use of the available domain knowledge [6, 190]. This is also reflected in the phrase "all models are wrong, but some models are useful", which is attributed to the statistician George Box [27]. Models are always "wrong" as they are only an abstraction of reality. However, if the model type, or inductive bias, is chosen properly, the resulting model can be useful in practice.

Performance Assessment

The performance measure P reports how well a model predicts the target features. Rather than maximizing the accuracy, regression algorithms usually minimize the prediction error [120]. The most commonly used error measure for regression tasks is the mean-squared error from Eq. 2.8. It averages the squared residual between the target value y_i and the predicted value $f(x_i)$ over the number of data samples N.

The *rooted mean squared error* (RMSE) is an alternative measure, which offers the advantage that the error has the same unit as the target variable.

$$RMSE = \sqrt{\frac{1}{N}\sum_{1}^{N}(y_i - \hat{y}_i)^2} \tag{2.13}$$

The *coefficient of determination* (R^2) describes how well a model predicts the response variable compared to a simple prediction equal to the mean value of the response variable.

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{n=1}^{N}(\hat{y}_n - y_n)^2}{\sum_{n=1}^{N}(\bar{y}_n - y_n)^2} \tag{2.14}$$

with $\bar{y}_n = \frac{1}{N}\sum_{n=1}^{N} y_n$ referring to the mean of the response variable and TSS is the total sum of squares. A perfect fit has a value of $R^2 = 1$, as the RSS will be equal to 0.

Data Handling, Generalization, and Overfitting

The process of learning a model from data is typically split into training and testing phases. To this end, the data is typically split into three distinct sets: train, validation and test. The train and validation sets are considered during the training phase, where the learning algorithm adapts the model to fit the observations from the training dataset. Simultaneously, the performance on the validation set is computed, with the primary purpose of preventing overfitting and ensuring generalization. Generalization is an important characteristic of a good model, since the goal is not to fit the training data perfectly, but to use the learned model for future predictions on input data distinct from the training data [237]. When the performance on the training and validation sets diverge, i.e., the training performance improves, but the validation performance deteriorates, this is usually a sign of overfitting to the training data [190]. While the training data is learned well, the performance on unseen samples is poor. In an extreme case, one could use a ten-degree polynomial to fit a curve through a dataset with $N = 10$ samples. While the training error would be equal to zero, the test error will most likely be high. Instead of understanding an underlying trend in the data, the model learns the data samples themselves. This is an undesired behavior because generalization suffers. As Fig. 2.1d indicates, the training error decreases with increasing polynomial degree. At the same time, the validation error increases, which indicates overfitting and bad generalization. For this specific example, a quadratic polynomial model would be the best choice. Once training and validation are finished, the test dataset, also referred to as the holdout set, is used as a benchmark to evaluate the predictive capabilities of the model on unseen data.

The Power of Visualization

Fig. 2.2 displays Anscombe's quartet, which is an often referred to set of four datasets which produce the same linear fit on different data distributions [190]. All datasets share the same statistics, i.e., mean values with respect to the x and y variables. While the linear regression model on the first dataset fits the data samples very well, the second dataset indicates how a straight line is not a good fit for the underlying nonlinear distribution. Datasets three and four demonstrate how a single data sample can influence the overall regression model, with the fourth dataset showing how a so-called outlier with a large distance to the other samples produces an unexpected result when the data is visualized.

This example demonstrates that, in many cases, pure statistical analysis is insufficient for producing a satisfactory fit of a regression model. It also illustrates the value of data visualization in providing insights into the underlying distributions.

Model Interpretability vs. Prediction Accuracy

In addition to good accuracy and generalization, the interpretability of a model is an important factor in practice [120]. In scientific research, regression methods are commonly used for model inference, i.e., the goal is to understand the

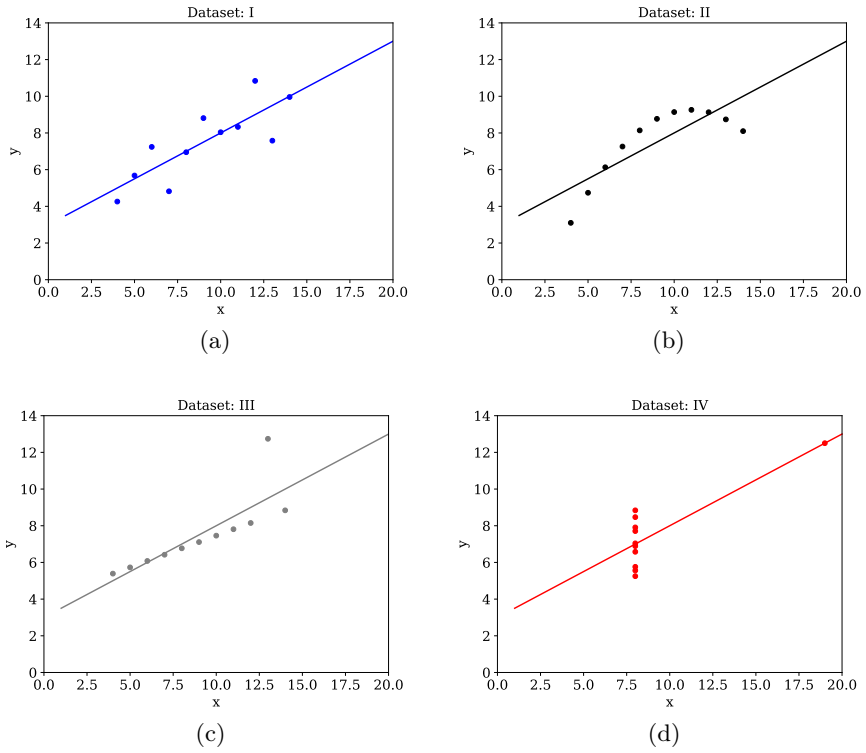


Figure 2.2: Anscombe's quartet shows how different data distributions create the same linear fit. Code to generate the plots inspired by the codebase [191] related to [190].

underlying relationships and causal mechanisms within the data. To this end, inferential models often prioritize simplicity and interpretability, rather than solely focusing on predictive accuracy. Intuitively, humans perceive simpler models as more interpretable [237]. However, more complex models tend to be more accurate, as the previous example with high-degree polynomials demonstrated. While the definition of simplicity is not straightforward, it can be argued that a tenth degree polynomial model is more complex than a simple linear model. Thus, if two models explain the data equally well and model m_1 is less complex than model m_2, the simpler model m_1 is preferred. This principle is also known as Occam's razor [190] (or Ockham's razor [237]). Simplicity is not only in the interest of interpretability, but also to avoid overfitting.

2.2 Artificial Neural Networks

ANNs have become a fundamental component of modern ML, due to their flexibility in terms of architectures and tasks that can be approached. Multilayer feedforward networks are regarded as universal approximators [117], as they can approximate any (highly nonlinear) underlying relation between variables with a certain precision, if the model architecture fits the complexity of the problem. Different architectures have emerged over time, such as *convolutional neural networks* (CNNs), which are particularly powerful at image classification [160], *recurrent neural network*s (RNNs) for processing sequential data [112], as well as *graph neural network*s (GNNs) [244], which operate on graph-like structured data. These, and even more advanced architectures, share the same fundamental principles, which will be introduced subsequently.

2.2.1 Multilayer Perceptrons

The Core Idea

As mentioned earlier, linear regression models have the disadvantage that they cannot fit any (non-linear) relationship between two or more variables. A potential way to overcome this issue is to apply a non-linear mapping $\Phi(X)$ to the input X rather than learning the model on X. This technique is applied

in the previously introduced SVR using the kernel $\mathcal{K}$, as well as nonlinear transformations of input variables as in Eq. 2.4. However, the choice of $\mathcal{K}$, or Φ in the general case, has a high influence on the performance. The idea of ANNs is to overcome this issue by learning Φ from a broad function family, so that

$$y = f(X; \theta, \omega) \tag{2.15}$$

where θ are the parameters to learn $\Phi = f(X, \theta)$, and the parameters ω map from Φ to the output y [98]. Although θ and ω have different roles theoretically, in practice, they are both learned together and play analogous roles to weights and biases in the network. For the sake of simplicity, we will conflate the parameters θ and ω as weights w and biases b, independent of the position within the network.

Network Structure

An MLP, as originally proposed by Rosenblatt [233], comprises an input layer, one or multiple hidden layers as well as an output layer, which are stacked upon each other. A *feedforward neural network* (FFNN) is a directed acyclic graph of nodes (or neurons) and edges between them. FFNNs predict the target variable by propagating inputs through the hidden layers to the outputs in a single forward pass, without any feedback loops from the outputs back to the inputs. In a *fully connected neural network* (FCNN), each neuron in the previous layer has an edge to all neurons in the next layer [1].

The number of layers $\ell \in \{1..L\}$ determines the depth of a network. Each layer ℓ consists of K_ℓ neurons, which is referred to as the width of a layer. The input to a neuron is a weighted sum of the outputs of the connected neurons from the previous layer. A bias term $b_{\ell k}$ can optionally be added to the weighted sum, which shifts the input sum by a constant value. Each neuron comprises an activation function φ, which applies a nonlinear transformation to the input [95]. Thus, the output of a neuron h located at a position k in a layer ℓ can be written as

$$h_{\ell k} = \varphi_\ell \left(b_{\ell k} + \sum_{j=1}^{K_{\ell-1}} w_{\ell k j} h_{\ell-1,j} \right) \tag{2.16}$$

where $w_{\ell k j}$ refers to the weight at the edge between the output of a neuron j in layer $\ell - 1$ and a neuron k in layer ℓ [190]. Fig. 2.3 illustrates how inputs are aggregated and passed through a general MLP architecture.

The mapping function Φ of an MLP thus can be decomposed into a stack of multiple simpler functions, so that

$$\Phi(X; \theta) = f_L(f_{L-1}(\dots f_1(X) \dots)) \tag{2.17}$$

where $f_\ell(X) = f(X; \theta_\ell)$ is the function at layer ℓ.

Types of Activation Functions

Introducing nonlinearities is a key feature of MLPs [1]. Different types of activation functions φ are used in practice, depending on the problem at hand. Common activation functions include the sigmoid function $\sigma(x) \in (0, 1)$[1], which scales values between zero and one. The hyperbolic tangent $\tanh(x)$ generated values $\tanh(x) \in (-1, 1)$. The *rectified linear unit* (ReLU) is a linear function for $x > 0$, and equal to zero for $x \leq 0$, so that $\mathrm{ReLU}(x) \in [0, \infty)$ [98].

1. Round brackets indicate an open interval, where endpoints are not included in the interval, and square brackets a closed interval with included endpoints.

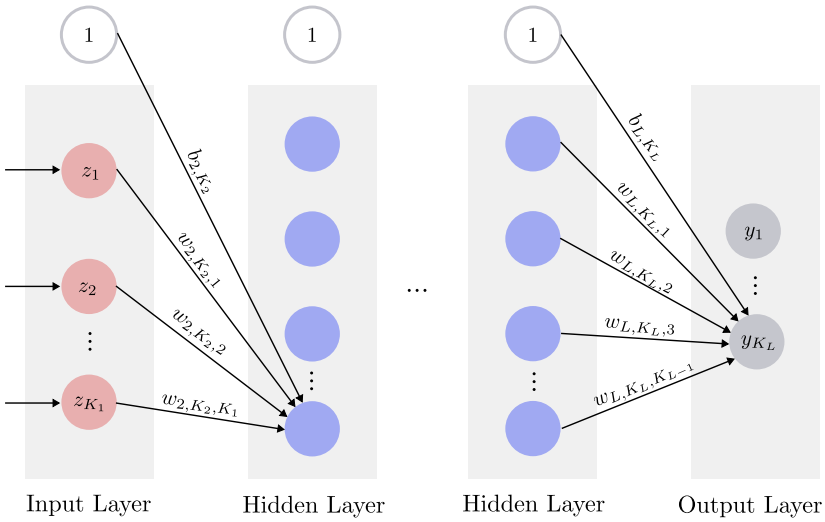


Figure 2.3: An MLP with full connections between all layers. For the sake of clarity, only the connections from the input layer to the last neuron in the first hidden layer, as well as the connections from the last hidden layer to the last output neuron are illustrated with their respective weights $w_{\ell kj}$ and biases $b_{\ell k}$. The subscripts refer to the neuron k within the layer ℓ, as well as the position j of the connecting neuron from layer $\ell - 1$. Blue neurons in the hidden layers apply an activation function φ.

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.18}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.19}$$

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.20}$$

Training with Backpropagation

The output of an MLP is computed in a single forward pass from the input layer, through the hidden layers, to the output layer. A cost function $J(\theta)$ considers the error between the predictions $\hat{y}$ and the target values y, as well as optional regularization of the parameters θ. During training, the parameters θ, constituting the weights w and biases b, of the MLP are adapted to fit the data points, i.e., minimize the cost function $J(\theta)$. The backpropagation algorithm was proposed in 1986 by [235] for efficient parameter adaptation. It involves the calculation of the gradient of the cost function $\nabla_\theta J(\theta)$ with respect to the parameters θ. The chain rule of calculus is recursively applied to propagate the gradient back through the network layers. Typically, gradient descent is used to learn from the gradient and update the parameter values [98]. A mini-batch version of GD is commonly applied, where predictions are made on a subset of the dataset, which has a batch size B. This is considered a good trade-off between the fast computations of *stochastic gradient descent* (SGD), which uses single data points, and batch training considering the entire dataset for one parameter update step. The overall algorithm thus constitutes a forward pass to make predictions and compute a cost function, a backward pass in reverse through all layers to measure the contributions of each parameter to the cost function, and using this information for parameter tweaking [95]. A more detailed formulation of the MLP training algorithm is provided in [98, Chapter 6].

Regression MLP Design Decisions

The architecture and design of an MLP determines how well the mapping $\Phi(X, \theta)$ can be learned to fit the data. Deeper networks with more hidden layers are capable of extracting more complex features from the data. Wider networks can be effective for problems where relationships in the data are less hierarchical and more parallel, allowing the network to capture various aspects of the data simultaneously. For regression tasks, the output neurons typically do not apply an activation function to avoid restricting the output value range. The MSE is usually used as the cost function, with optional regularization terms as introduced in Sec. 2.1.4. Further design decisions include the learning rate η, which determines the which determines the step size for each iteration during the optimization process and the batch size B of mini-batch training. A non-exhaustive list of common issues in training MLPs includes vanishing and exploding gradients, overfitting and underfitting, as well as working with noisy

and imbalanced data. They can be addressed with further tuning techniques such as sophisticated weight initialization, batch normalization, additional regularization methods, adaptive learning rate methods, dropout, early stopping, or data augmentation. These are explained in more detail in [22, 98, 95].

2.2.2 Graph Neural Networks

Graphs are an omnipresent data structure and can be used to describe various systems and relations between entities. GNNs are specifically tailored to handle data represented in graph structures, where nodes represent entities and edges represent relationships between them. GNNs have gained significant attention recently for their effectiveness in tasks involving relational data, such as social network analysis, recommendation systems, and molecular property prediction.

Graph Theory

A graph $G(V, E)$ consists of a non-empty finite set of vertices $V(G)$, also referred to as nodes, and edges $E(G)$ connecting the vertices. An edge $\{v, v'\}$ is represented as a pair of elements of $V(G)$. For undirected graphs, the pair of vertices is unordered. To the contrary, directed graphs contain an ordered pair of vertices to define an edge from a vertex v to a vertex v' [293]. While simple graphs only allow one edge between two distinct vertices, multi-graphs can include self-loops, as well as multiple edges between the same pair of vertices. Weighted graphs have a weight or cost assigned to each edge, denoted by $w(e)$. The degree of a vertex $deg(v)$ counts the number of edges connected to that vertex. For directed graphs, the number of ingoing edges is defined as $deg^-(v)$, and the outgoing edges as $deg^+(v)$. A common representation of the edges of a graph is the adjacency matrix. For a graph with n nodes, the adjacency matrix is a square matrix of size $n \times n$. An entry of 1 at a position i, j in the matrix indicates an edge between the vertices i and j, 0 indicates no edge. As this representation can grow huge with many vertices in a graph, the edge list is a more concise representation of the same. It is a list of vertex pairs that are connected by an edge, which is especially useful to represent sparse graphs with few edges [155].

Graph Structures for Neural Networks

Different types of models from the GNN family have been explored in the past [e.g. 19, 99, 244]. The *message passing neural network*s (MPNNs) framework as introduced by Gilmer et al. in 2017 consolidates some previous GNN architectures [96]. Battaglia et al. further generalize and extend this framework for applicability to a broader range of domains [18]. We use their definition of MPNNs in this thesis.

The general GNN block as proposed in [18] is based on directed multi-graphs with attributes at different levels. The previously introduced definition of a graph is extended to a 3-tuple $G = (\mathbf{u}, V, E)$. It constitutes the following elements:

- $\mathbf{u}$ represents global or universal properties. Using the example of a mass-spring system, where each mass object is modeled as a node, and the connecting springs as edges, a universal property can be the gravitational field in which the system is placed.
- $V = \{\mathbf{v}_i\}_{i=1:N^v}$ is the set of nodes, which describe entities. The total number of nodes in the graph is N^v. $\mathbf{v}_i$ are the node features, which describe properties such as mass, position, or velocity of the mass objects in the mass-spring system.
- $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:N^e}$ is the set of edges, describing relations between entities. N^e represents the number of edges in the graph. $\mathbf{e}_k$ contains the edge features, such as the spring constant of the connecting springs in the mass-spring system. An edge connects the receiver node with the index r_k to the sender node s_k.

The generalized GNN block is flexible in terms of its structure and function configurations so that it can be adapted to different purposes. In this thesis, we are mainly concerned with MPNNs.

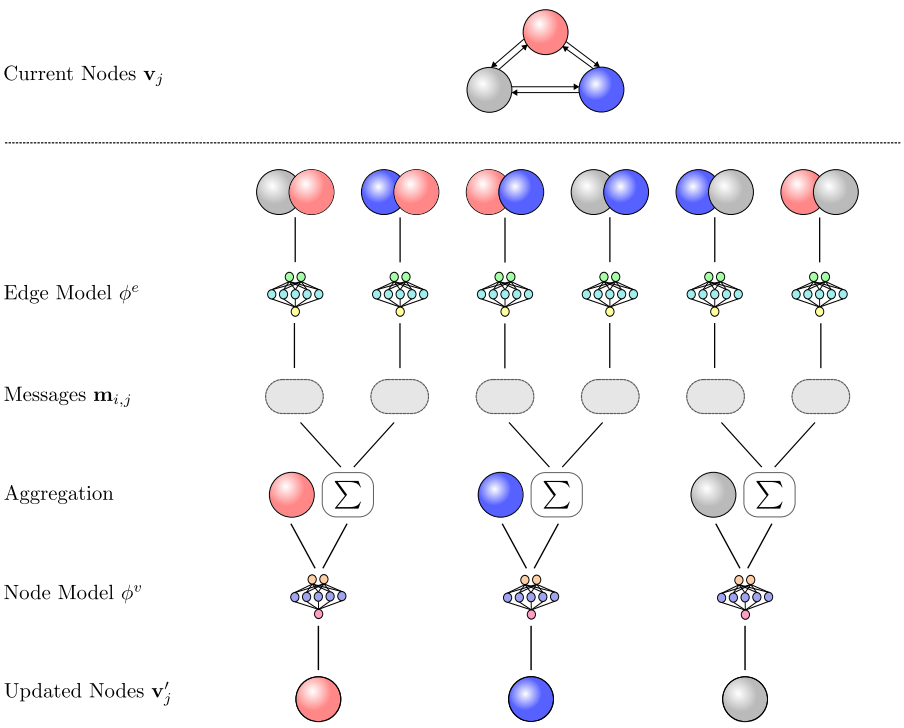


Figure 2.4: One update step of an MPNN on a graph with three nodes and mutual neighboring connections. Figure inspired by a similar illustration in [60].

Message Passing Neural Networks

The main idea behind message passing is to iteratively update the representation of each node by aggregating information transferred through messages from its neighboring nodes. This approach allows each node to incorporate local structural information and features from its surrounding nodes, enabling the network to learn complex dependencies and patterns within the graph. It does that by performing consecutive steps of message computation, aggregation, and node update.

$$\mathbf{m}_{i,j} = \phi^e(\mathbf{v}_i, \mathbf{v}_j, \mathbf{e}_{i,j}) \qquad \textit{message computation} \qquad (2.21)$$

$$\mathbf{m}_j = \rho^{e\to v}(\{\mathbf{m}_{i,j}\}_{i\in\mathcal{N}(j)}) \qquad \textit{message aggregation} \qquad (2.22)$$

$$\mathbf{v}'_j = \phi^v(\mathbf{v}_j, \mathbf{m}_j) \qquad \textit{node update} \qquad (2.23)$$

$$\bar{\mathbf{v}}' = \rho^{v\to u}(\{\mathbf{v}'_i\}_{i=1:N^v}) \qquad \textit{node aggregation} \qquad (2.24)$$

$$\mathbf{u}' = \phi^u(\bar{\mathbf{v}}') \qquad \textit{universal property update} \qquad (2.25)$$

The feature vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ of nodes i and j, as well as the feature vector $\mathbf{e}_{i,j}$ of the edge between nodes i and j are the input to the edge model ϕ^e. It computes a message vector $\mathbf{m}_{i,j}$ sent from node i to node j in Eq. 2.21. The incoming messages of a node j, sent from nodes in its neighborhood $\mathcal{N}(j)$, are aggregated in Eq. 2.22 using the aggregation function $\rho^{e\to v}$. In MPNNs, typically elementwise summation is applied, so that $\rho^{e\to v} = \sum_{i\in\mathcal{N}(j)} \mathbf{m}_{i,j}$. The aggregated message vector $\mathbf{m}_j$, as well as the current node state $\mathbf{v}_j$, are the input to the node model ϕ^v to compute the updated node $\mathbf{v}'_j$ (Eq. 2.23). The aggregated global node feature vector $\bar{\mathbf{v}}'$ is obtained by aggregating the updated feature vectors of all nodes in the graph (Eq. 2.24). These are used to compute the updated global property $\mathbf{u}'$ using the global model ϕ^u (Eq. 2.25). They represent features or attributes that describe the entire graph.

For many systems, including the ones addressed in this thesis, computations on the edge and node level suffice, i.e., the update step is complete once Eq. 2.23 is executed. Fig. 2.4 illustrates the principles of MPNNs using the example of a graph with three nodes and mutual pairwise interactions. Both ϕ^e and ϕ^n use shared parameters for all message computations and node updates.

2.3 Evolutionary Algorithms

The field of *evolutionary computation* (EC) unites nature-inspired algorithms, with the two main categories of *evolutionary algorithms* (EAs) and *swarm intelligence* (SI). SI algorithms are inspired by the self-organized, decentralized collective behavior which is often encountered in swarm-like structures in nature, such as birds flocks, ant colonies or fish schools. Popular algorithms, for instance particle swarm optimization [129] and ant colony optimization [72], have emerged from this field. EAs make use of the Darwinian principle of survival of the fittest, and mimic evolution to find solutions to optimization and search problems [147]. Multiple subcategories of EAs with different solution representations each have evolved, namely *evolutionary programming* [89], *evolution strategies* [20], *genetic algorithms* [113], and *genetic programming* [140]. The main topic of interest in this thesis is genetic programming, which stems from the family of EAs. We will introduce the general principles of evolutionary algorithms and the related field of multi-objective optimization, before deriving the principles of genetic programming.

2.3.1 General Principles

Metaheuristics

Evolutionary algorithms are optimization techniques that belong to the family of metaheuristic methods. Metaheuristics are applied when there is no effective solution to a complex problem. These methods typically do not find exact solutions due to the high demand of computational time and power induced by the complexity of the problem. Instead, they approximate "good enough" solutions that can be obtained within acceptable computational time and power. Although numerous techniques exist, all metaheuristics share a common underlying principle: the incorporation of random elements, coupled with the implementation of certain guidance measures that direct the solution space, based on the quality of the available solutions. In this way, they balance exploration, i.e., diversity in the solution space, and exploitation, which leverages known solutions to improve them further [147, Chapter 11.1].

Algorithmic Procedure

The core idea of EAs is that an initially randomly generated population of μ solutions improves its quality by imitating evolutionary behavior over several iterations. First, an initially random population of individuals is sampled from the search space. Their quality is evaluated using a problem-dependent fitness function. New solutions are created by selecting the best solutions per generation, the so-called mating selection, and applying genetic operators in a recombination procedure to obtain λ children. Individuals with higher fitness are preferred, following the assumption these have valuable characteristics that are well adapted to the environment and worth passing on to the next generation. In the (μ, λ) reproduction scheme, only children survive to the next generation, whereas the $(\mu + \lambda)$ reproduction scheme considers both children and their parents from the previous generation. This step is called the survival selection. The selected individuals build the new population for the next iteration, or generation in biological terms. The process is repeated starting from the mating selection step until a stopping criterion is reached.

Components of an EA

All EA methods share a number of common high-level components, which may vary in their implementation depending on the specific method in question [147]. Being a population-based method, the goal is to evolve a population of solution candidates, for which a problem-dependent and appropriate *encoding* is required. Encodings can take on arbitrary shapes, for instance real-valued vectors, or mathematical trees as for *genetic programming* (GP). Using the bio-inspired terminology, a solution candidate is also referred to as an *individual*, and the encoding as the *genotype*. The *phenotype* is derived from the genotype and is the actual solution to the problem, i.e., how the genotype is interpreted in the problem domain. An *initialization procedure* is required to generate the initial set of solutions in the population, which are randomly generated

sequences in the simplest case. However, due to constraints on solution properties such as the length or order of values, more sophisticated initialization procedures may be necessary for certain problems. Problem-dependent one or multiple *fitness functions* emulate the influence of the biological environment and provide information about how well an individual is adapted to this environment, i.e., to assess how "fit" it is. Often, a fitness function is identical to the optimization function of interest. For some problems, it might be beneficial to consider additional criteria, such as the extent to which constraints are fulfilled. A typically fitness-based *selection method* is applied to identify individuals at two stages: first, to select parents for the subsequent mating process, and second, to identify individuals who survive to the next generation. The *genetic operators* are employed to modify selected solutions and thereby build new solutions. *Mutation* operators introduce slight random perturbations into an existing solution, while *crossover* exchanges randomly selected parts of each parent with each other. While the algorithm could theoretically continue forever to evolve new individuals, in practice, a final solution to the problem is required, which implies the need for a *termination criterion* for the search. Potential criteria involve a predefined number of algorithm iterations, so-called *generations*, stagnation of fitness improvement for a certain amount of time, or when a user-defined fitness value is reached. The final component of an EA are problem and algorithm dependent *parameter values*, such as population size or crossover and mutation probabilities.

Use Cases and Limitations

Other than classical optimization methods which seek an exact analytical solution or are gradient-based (see Sec. 2.1.2), EAs usually optimize gradient-free and without imposing restrictions such as requiring the function to be unimodal. In this way, EAs can handle issues like non-differential functions and multimodality, where other optimization methods reach their limits. The population-based approach allows searching multiple regions of the search space simultaneously, which reduces the influence of the initial parameter values and helps to avoid convergence to local optima [66]. While EAs incorporate multiple components to overcome issues of classical optimization methods, they, of course, have certain limitations themselves. First, EAs have no guarantee to find the best solution, but are mainly designed to find practically useful solutions within a reasonable amount of time. Furthermore, due to their stochastic nature involving randomness, the convergence is not guaranteed and results differ when the algorithm is repeated multiple times. In addition, EAs are sensitive to the choice of the parameter values and the encoding, both of which are highly problem-specific, making it difficult to provide general guidelines. When these limitations are addressed through careful parameter tuning, for instance by incorporating domain knowledge, EAs are a powerful and adaptive optimization method.

2.3.2 Multi-Objective Optimization

Due to the simultaneous exploration of several areas within the search space, EAs are well suited to optimizing not only one, but multiple criteria at the same time. Rather than finding one optimal solution for each criterion, the goal is to find a set of solutions that satisfy the criteria to different extents. While simple methods like weighted sum combine the different criteria into one measure, the focus of this thesis lies on *multi-objective optimization* (MOO), where the criteria (also referred to as objectives or fitness functions) are kept separate. Looking at optimal solutions in this way furthermore requires a redefinition of optimality, as well as search algorithms that can handle multiple objectives [66, 97].

Optimization Problems

A general optimization problem consists of a pair (Ω, f), where Ω constitutes the search space of all potential solutions to the problem, and f is an evaluation function $f : \Omega \to \mathbb{R}$ that maps a candidate solution $\omega \in \Omega$ to a fitness value $f(\omega)$ [147, Chapter 11.3.1]. Building upon this notation, the formal definition of a *multi-objective optimization problem* (MOOP) with m conflicting objectives

is denoted by

$$\begin{aligned} \min / \max \quad & \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x}))^T \\ \text{s.t.} \quad & \mathbf{x} \in \Omega \end{aligned} \tag{2.26}$$

where $\mathbf{x}$ represents a solution in the search space Ω and $f_i(\mathbf{x})$ the fitness value in the i^{th} objective, with $i = 1, 2, \ldots, m$ [66]. Typically, a problem is formulated to either maximize or minimize all objectives. In this thesis, all objectives are minimized, and maximization objectives are transformed into minimization objectives by employing the negative value of the function.

Dominance and Pareto-Optimality

As soon as multiple objectives are employed, no strict ordering within a set of solutions according to a single objective is given. The concept of Pareto dominance plays a crucial role, identifying solutions that represent the best trade-offs among the multiple objectives. In the context of optimization, Pareto-optimality applies to a solution where an objective value cannot be improved without simultaneously deteriorating at least one other. A solution $\mathbf{x}_1$ Pareto dominates another $\mathbf{x}_2$ (denoted by $\mathbf{x}_1 \prec \mathbf{x}_2$) if and only if the following two criteria are fulfilled [66]:

1. $\mathbf{x}_1$ is no worse than $\mathbf{x}_2$ in all objectives, i.e., $f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2)$ for all $i = 1, 2, \ldots m$.
2. The solution $\mathbf{x}_1$ is strictly better than $\mathbf{x}_2$ in at least one objective, i.e., $f_i(\mathbf{x}_1) < f_i(\mathbf{x}_2)$ for at least one $i = 1 \ldots m$.

This results in the mathematical definition of Pareto dominance as

$$\begin{aligned} \mathbf{x}_1 \prec \mathbf{x}_2 \iff & (\forall i \in \{1, \ldots, m\}, f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2)) \\ & \wedge (\exists j \in \{1, \ldots, m\}, f_j(x_1) < f_j(\mathbf{x}_2)) \end{aligned} \tag{2.27}$$

The Pareto-optimal set P^* is the set of all non-dominated solutions in the search space Ω:

$$P^* = \{x \in \Omega \mid \nexists x' \in \Omega \text{ such that } x' \prec x\} \tag{2.28}$$

And the corresponding Pareto front PF^* is the image of the Pareto-optimal set in the objective space:

$$PF^* = \{f(x) \mid x \in P^*\} \tag{2.29}$$

Fig. 2.5 illustrates an example of a Pareto-optimal front and Pareto-dominance.

Non-Dominated Sorting Algorithm

Several *multi-objective evolutionary algorithms* (MOEA) have been developed over time to address multiple objectives in evolutionary optimization. The goal is to identify various Pareto-optimal solutions which cover the *Pareto-optimal* (PO) front. Technically speaking, an MOEA maps the n-dimensional search space Ω to the m-dimensional objective space $\mathcal{M}$. Next to the *strength Pareto evolutionary algorithm* (SPEA) [300], the *non-dominated sorting algorithm* (NSGA) [261] was one of the earliest algorithms proposed to solve this problem. The improved version NSGA-II was presented in 2002, which to date can be considered one of the most influential and widely used algorithms for multi-objective optimization [65]. NSGA-II divides the population into multiple non-dominated fronts by iteratively finding the front of the current population, storing it in a separate set, removing the individuals of this front from the population and increasing the front index by one. This process

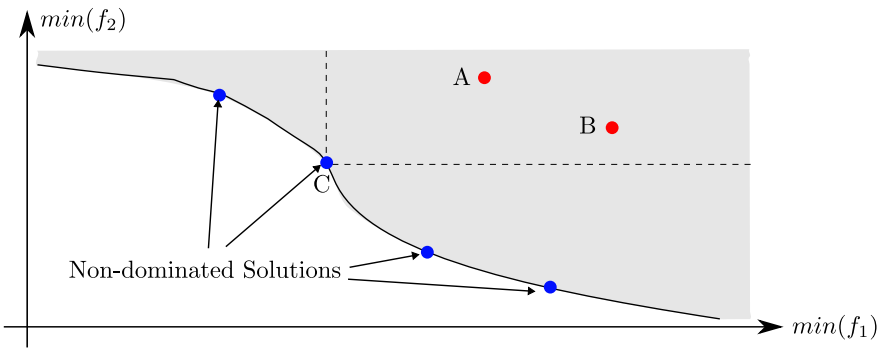


Figure 2.5: Pareto front of a two-objective problem with both objectives to be minimized, where solution C Pareto-dominates the solutions A and B.

is called non-dominated sorting and is repeated until all individuals belong to a front set. All individuals of a set have the same fitness value equal to the front rank, where low fitness is generally better. The new population is then filled up with individuals from the front sets until adding the next front would exceed the population size μ. Here, *crowding distance* (CD) comes into play, which is applied to select a uniformly spread-out set of solutions among the current non-dominated front. Solutions with higher CD values are located in less crowded areas and are preferred over the ones in crowded areas to maintain diversity in the objective space. NSGA-II is a universal algorithm that also finds application in multi-objective genetic programming, which is the main concern of this thesis. The original algorithm as well as the pseudocode for non-dominated sorting and crowding distance have been introduced in [65].

2.4 Genetic Programming

The following section is largely based on the author's publication [221].

GP is an EA technique first introduced by Koza in 1992 to evolve computer programs of variable size and shape [140]. Many solutions for ML problems can be naturally represented as a hierarchical composition of primitive functions and terminals, such as computer programs, algorithms, and symbolic expressions. GP employs evolutionary principles to identify and construct such hierarchical structures automatically, by providing examples of the desired behavior or output and tasking the algorithm with the explicit generation of the program. It is a common technique used to address SR problems, which involve the automated development of free-form mathematical equations. The limitations of classical regression algorithms are circumvented by making the discovery of the appropriate functional form part of the learning task. While GP for regression tasks is the main focus of this thesis, it is important to note that there exist numerous other applications where GP is a suitable approach, such as classification, scheduling, and control [292]. In the following, an overview about GP and its specific characteristics for symbolic regression will be given.

2.4.1 Canonical GP Algorithm

Borrowing the concepts of Darwinian evolution, the canonical GP algorithm operates on a population of expression trees, which are iteratively refined by selecting fitting individuals and applying genetic operators. Algorithm 1 outlines the basic steps to learn a hierarchical structure from the training data X using the $(\mu + \lambda)$ reproduction scheme with combined parent and child populations. Crucial hyperparameters of a GP algorithm are the population size μ, the number of children λ, as well as the crossover and mutation probabilities p_c and p_m respectively, which determine whether a child is generated through crossover or mutation on the parents. The remainder of this section will provide a more detailed examination of the fundamental elements of the GP algorithm.

Algorithm 1: Canonical Genetic Programming Algorithm

input : Training Data X, Population Size μ, Number of Children λ, Crossover Probability p_c, Mutation Probability p_m
output: Solution Population P

$P \leftarrow$ initializePopulation(μ) `// create random initial population`
evaluate(P, X) `// compute fitness values`
while not termination criterion **do**
  $P_{\text{child}} \leftarrow \emptyset$ `// create empty child population`
  for $i = 1 \dots \lambda$ **do**
    $genOp \leftarrow$ selectGenOp(p_c, p_m) `// select genetic operator`
    $parent \leftarrow$ selectParents(P) `// select parent(s)`
    $child \leftarrow$ genOp($parent$) `// apply genetic operator`
    $P_{\text{child}} \leftarrow P_{\text{child}} \cup child$ `// add individual to child population`
  end
  evaluate(P_{child}, X) `// evaluate child population`
  $P \leftarrow$ environmentalSelection($P \cup P_{\text{child}}$) `// survival selection`
end
return P

2.4.2 Representation

Function and Terminal Sets

Formally, GP requires a tuple of sets $(\mathcal{F}, \mathcal{T})$, where the function set $\mathcal{F}$ contains all function symbols and operators and the terminal set $\mathcal{T}$ entails all input variables and constants, so-called terminals. Together, the function and terminal set build a so-called primitive set. Selecting an appropriate set of primitives that is sufficient to express a solution to the given problem is a non-trivial task and highly problem dependent. Typical functions in the function set can include arithmetic, mathematical, boolean as well as conditional operations, with a non-exhaustive list includes

$$\begin{aligned}\mathcal{F} = \{&+,\ -,\ \cdot,\ /,\ \cos(\circ),\ \sin(\circ),\ \tan(\circ),\ \exp(\circ),\ \log(\circ),\\ &\wedge,\ \vee,\ \text{IfThenElse},\ \dots\}\end{aligned} \tag{2.30}$$

where $\circ$ is a placeholder for function input. The arity of a function determines the number of input variables, i.e., two for binary and one for unary functions [140, 147]. The function set can be expanded to include arbitrary functions and an unlimited number of input arguments, which renders GP a flexible approach that can be adapted to a diverse range of problem domains. As the definition of the function set is highly problem-dependent, it should satisfy the two properties of *sufficiency* and *closure*. Closure is the property ensuring that all functions and terminals in a GP tree can accept any arguments they might receive, and sufficiency is a property that aims to identify suitable solutions by ensuring that the two sets possess enough expressivity. Sufficiency generally cannot be guaranteed, but prior knowledge about potential solutions to related problems can support the selection of an appropriate function and terminal set that is likely to contain components capable of constructing effective solutions [210]. Using the function and terminal sets, the search space of a GP algorithm is the set of all possible functions that can be constructed from these primitives. Due to the enormous number of functions with increasing number of primitives, the search space is usually restricted by the definition of a maximum tree depth or number of nodes [75].

Tree Representation

The syntax tree is the most commonly used representation and became prevalent since it is easily interpretable and can be parsed into an executable program or equation. For the computational implementation, i.e., genotype of a syntax tree, a complex tree data structure or a simple list data structure can be used. The latter is less computationally expensive to process, while the former can be more easily visually interpreted by a user. Syntax trees overcome the fixed-

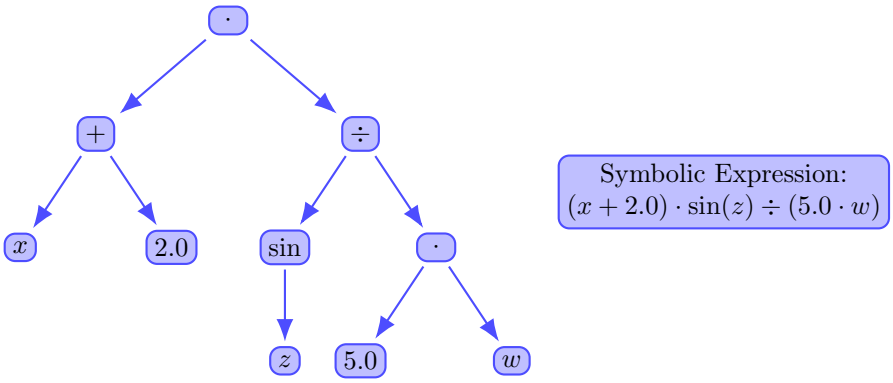


Figure 2.6: Syntax tree representation of a symbolic expression.

length encodings of EAs and allow for various functional forms. This thesis employs the syntax tree as the primary representation for GP solutions, as it is the most prevalent representation in the literature and software frameworks.

Other Representations

Further representations include linear GP techniques, such as grammatical evolution [194, 238], gene expression programming [84], linear genetic programming [28], as well as Cartesian genetic programming [178, 179]. A comparison of these can be found in [197]. Prominent extensions of the tree-based representation are strongly-typed GP [183], which enforces data-type constraints, as well as grammar-guided GP [175, 290], which overcomes the closure requirement by using grammars to specify valid syntax and structure for solutions. This ensures that all generated programs are syntactically correct and relevant to the problem domain, which is especially relevant in the phases of initialization and application of genetic operators.

2.4.3 Initialization

Importance of Initialization

Initialization plays a crucial role in the evolutionary process, as the genetic material introduced in the initial population forms the foundation of the search for optimal solutions. The initial population is essential for diversity and exploration capabilities within the search space. While mutation introduces random changes and can insert new genetic material, crossover operations can only manipulate and combine existing genetic material, thus heavily relying on the initial genetic material. Typically, the initial population is created randomly, but guided by predefined criteria such as an upper limit for tree depth or number of nodes.

Initialization Methods

The goal of a well-designed initialization method is to ensure diversity of solutions in terms of structure, utilized operators, and combinations of operators and variables. Several strategies exist for initializing the population. The *full* method always generates fully balanced trees with a fixed depth. The insertion of terminal nodes is only allowed at the last level of the predefined tree depth. The *grow* method, to the contrary, starts by producing trees with a minimum depth, and beyond that selects the next nodes from both the function and terminal sets. In this way, solutions of varying functional structures and terminal nodes at different levels are created [140]. *Ramped half-and-half* is another popular approach, where half of the initial individuals are generated using the *full* method with a predefined maximum depth, and the other half using the *grow* method. Various adaptations of these methods exist in the literature, for instance additional control over the likelihood with which functions and terminals appear in the equations [167], and seeding the initial population with functional terms that are expected to appear in a solution [158]. This can be especially useful when domain knowledge about potential solutions is available.

2.4.4 Fitness-Based Selection

Selecting fitting individuals is a driving force behind evolution, following the principle of survival of the fittest. The individuals are evaluated on problem-

dependent fitness functions and selected according to their fitness values.

Fitness Evaluation

For regression tasks, the main optimization objective is to minimize the error between the target variable and the prediction made by the expression generated by the algorithm. Common fitness functions for regression tasks are closely related to the cost functions in ML, which have already been introduced in Sec. 2.1.2. The most prominent fitness functions used in GP for SR include RMSE (Eq. 2.13), *mean absolute error* (MAE) (Eq. 2.9) and *mean squared error* (MSE) (Eq. 2.8), as well as various adaptations of the same containing regularization terms. A frequently used second optimization criterion for SR tasks approached with GP is the minimization of the solution complexity, according to the Occam's razor introduced in Sec. 2.1.4. The solution complexity is usually determined by counting the number of nodes in the tree. Individuals with higher fitness values are more likely to be selected in either of the following selection steps of the algorithm.

Single-Objective Selection

When solely the error is minimized as a single objective, the established single-objective selection methods from the family of evolutionary algorithms are employed, as the selection operation relies exclusively on the fitness values independent of the encoding or representation of a solution. Selection mechanisms such as tournament selection, roulette wheel selection, or rank-based selection are typically employed to choose the fittest individuals [147]. Further selection methods exist which are specifically tailored to GP. A comprehensive overview of GP selection mechanisms and their influence on the optimization process is given in [295].

Multi-Objective Selection

The selection step requires to be adapted when multiple objectives are optimized simultaneously, as a strict linear ordering between solutions is no longer possible. GP algorithms often balance accuracy and complexity, and thus need to consider both or even more objectives in the selection process. Tournament selection with the Pareto dominance criterion as quality measure for GP was introduced by [78] in 2007. Furthermore, the previously introduced concept of Pareto dominance combined with NSGA-II can be employed to compare solutions and sort them into different fronts. Individuals in lower fronts (with better trade-offs) are preferred, and diversity is maintained using measures like crowding distance [65]. Other methods include the SPEA-based selection as well as a Pareto-converging GP [297].

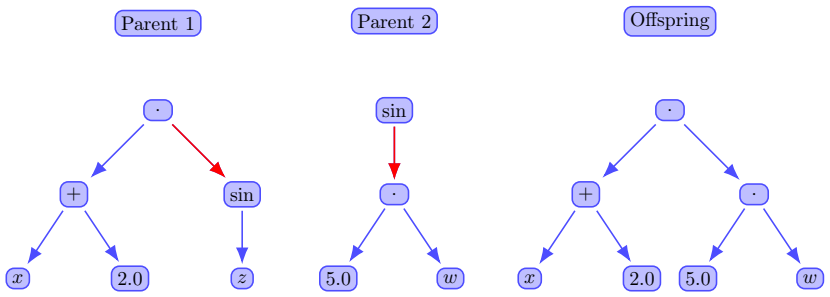
2.4.5 Genetic Operators

In addition to the selection step, one of the primary drivers of evolution is the application of genetic operators to generate new individuals from the selected parents. Genetic operators allow for a guided exploration of the search space towards better individuals, as they follow two main principles: First, strong individuals can benefit from recombination with other good solutions through crossover. Second, the likelihood of encountering further good solutions within the neighborhood of a strong individual is high, usually referred to as mutation. In order to perform these modifications on trees, various adaptations to the general EA reproduction are required.

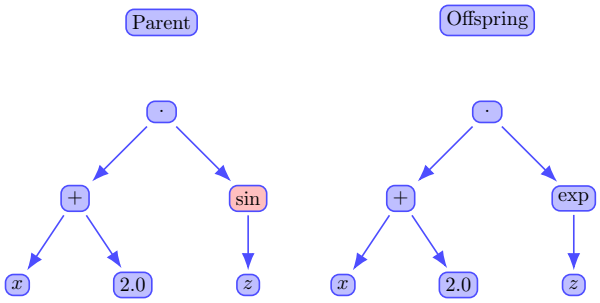
Crossover

For (one-point) crossover, two parents are required to produce a child with their combined features, as visualized in Fig. 2.7a. To this end, a random crossover point in each parent individual is selected. The subtree of the first parent originating from this cut-off point is then replaced with the subtree of the second individual. The solution emerged from this step is copied into the child population, while the parents remain untouched in the parent population. This enables individuals to participate multiple times in a reproduction operation with different partners [210]. Depending on the distribution of terminals and functions, the probability of selecting terminals as crossover points can be incomparably higher, which leads to the exchange of only a small amount of genetic material. To counter this, Koza suggested introducing an additional probability parameter that allows for steering the cut-off point selection [140].

Figure 2.7: Examples of crossover and mutation operation, where crossover points are marked as red arrows and mutation points as red nodes.



(a) One-point Crossover



(b) Point Mutation

Mutation Mutation is performed on only one parent individual to stay in the neighborhood of this individual. An example of a mutation operation is depicted in Fig. 2.7b. The two prevalent operations are subtree mutation and point mutation. As the name already implies, subtree mutation performs changes on a subtree of an individual. To this end, a cut-off point is randomly selected, and the attached subtree replaced by another randomly created subtree of user-defined depth [210]. This procedure can also be implemented as a one-point crossover between the selected individual as first parent and another randomly generated genetic program [8]. Commonly, subtree mutations are restricted to only replacing small subtrees with subtrees of a similar depth. Otherwise, the root node of a tree could be selected as the cut-off point, resulting in an entirely new random individual that is added to the population [147]. Point mutation, on the other hand, focuses on adding genetic diversity by modifying single nodes within an individual. These are randomly selected and in the further course replaced with a random primitive of the same arity. Depending on the implementation, point mutation is executed on either one or multiple nodes of a solution [210].

2.5 Summary of this Chapter

In this chapter, the scientific foundations on which the topics of this thesis are based were introduced. An overview of ML concepts and tasks was given, with a focus on supervised ML and regression, which is the main concern of this thesis. Classical regression methods and their approaches were presented. Choosing an appropriate regression model is a non-trivial task, and often prior problem knowledge is required to match the data well. Symbolic regression goes beyond the parameter fitting of classical regression methods and produces free-form equations from data. The basic idea of SR was introduced and general design decisions for regression algorithms were discussed. The basic concepts of ANNs and GNNs were presented, with a focus on the properties of MPNNs. Moreover, the general principles of optimization using evolutionary algorithms, and in particular MOO, were presented. GP is a method for SR stemming from the family of EAs, which is the main focus of this thesis. The canonical GP

algorithm and its tree representation for symbolic models were introduced, upon which the algorithms presented in this thesis were built.

3 Related Research and State of the Art

This chapter presents and discusses the related research that has been proposed in the literature and that is relevant to this thesis. The main focus is on *physics-informed machine learning* (PIML) and *genetic programming* (GP) for *symbolic regression* (SR).

Genetic Programming for Symbolic Regression

SR is a supervised machine learning approach aimed at identifying mathematical expressions that best fit a given dataset. Sec. 2.1.3 introduced the general idea of SR as well as some historical context. Compared to traditional regression methods, SR involves a larger search space, which slows down the search process. SR was recently classified as an NP-hard problem, i.e., there is no known algorithm that can solve all instances of the problem efficiently in polynomial time, due to the complexity of identifying optimal solutions [282]. Heuristic methods, particularly GP, have been extensively used to guide the search process. Recently, SR has attracted increased research interest, leading to the exploration of both *evolutionary computation* (EC) methods such as GP, as well as non-EC-based methods.

Chapter Structure

The integration of physical laws in machine learning algorithms has gained importance in the last few years, which is also relevant for this thesis as it addresses problems from the science and engineering disciplines. This ensures that models are not just learning patterns from data but are also guided by well-established physical theories and constraints. Sec. 3.1 covers the area of PIML, with a focus on *physics-informed neural network*s (PINNs) and physics-inspired SR methods outside the EC area. GP-related issues that are of particular interest for this thesis will be discussed in Sec. 3.2, together with methods developed in the past years to resolve these issues. Sec. 3.3 addresses the inclusion of domain knowledge in GP algorithms, which works hand in hand with the integration of physics into the *machine learning* (ML) process. In general, many individual approaches or examples of integrating problem-specific knowledge into GP algorithms have been presented in the literature. The literature analysis in this section gives a comprehensive overview of the related work landscape and classifies approaches according to how the algorithm is biased or constrained by domain knowledge. Finally, Sec. 3.4 summarizes this chapter.

3.1 Physics-Informed Machine Learning

Increased Interest in Domain Knowledge for Machine Learning

Lately, there has been a notable increase in interest regarding the integration of domain knowledge into data-driven modeling [13]. Previous machine learning research concentrated on developing algorithms that were largely independent

of domain knowledge. However, in the context of scientific and engineering applications, a model that is independent of domain knowledge may not necessarily reflect the expected behavior or adhere to the necessary physical rules, creating the need for domain knowledge constraints. This research direction is commonly referred to as "physics-inspired machine learning" or "physics-informed machine learning" in academic circles [13]. In this way, certain limitations inherent to data-driven modeling techniques can be circumvented, such as poor generalization on small or noisy datasets, lack of interpretability as well as scalability issues. As a result, there has been a significant increase in the number of publications dedicated to this research area, which also had a notable impact on the areas of neural networks and symbolic regression.

3.1.1 Neural Networks for Physics

Several *artificial neural network* (ANN) architectures have been developed in the last few years [161], some of which were specifically tailored to meet physics constraints. Covering the entire field of neural networks for physics tasks is nearly impossible and far beyond the scope of this thesis. In the following, a few approaches will be discussed which are interesting for the remaining course of this thesis. A detailed review of ANN approaches for physics can be found in [126].

Physics-Informed Neural Networks as Learning Bias

Raissi et al. [214] pioneered the use of PINNs. An extensive overview of the current state of the art of PINNs and their application areas was given by Cuomo et al. in [61]. The high-level idea of PINNs is the integration of physical domain knowledge as soft constraints on the loss function used by the machine learning algorithm. Specifically, it incorporates *partial differential equation*s (PDEs) of known physics in the loss function through automatic differentiation to emphasize solutions that satisfy physics constraints. The loss function thus constitutes two weighted parts, a residual from the original data and a residual on the PDEs. This method performed well in learning the flow wake past a cylinder in [215], with the Navier-Stokes Equations as additional PDEs. As pointed out in [143], the regular PINN approach showed ill-conditioned behavior for increasing problem complexities and a harder to optimize loss landscape. Thus, PINNs is a field of ongoing research, with various improvements having been proposed in terms of architecture and specific problem domains, such as meta-learning to accelerate the training of PINNs [21], and Δ-PINNs for complex geometries [241]. Integrating physics into a model through a soft constraint learning bias as defined in [126] was effective for a wide range of applications. A comprehensive overview of recent methods to integrate physics knowledge into neural networks, which goes beyond PINNs, was given in [288].

Other Network Architectures

Various network architectures have been designed to embed and strictly satisfy the prior assumptions about physics in the model. Ling et al. [166] proposed a multiplicative layer next to the output layer of an ANN to embed Galilean invariance for a fluid mechanics problem. *Hamiltonian neural networks* (HNN) [100] and *Lagrangian neural networks* (LNNs) [59] were used to learn dynamical systems and integrate conservation laws in the model. *Graph neural network*s (GNNs) [19, 30] provided a well motivated inductive bias for problems that can be modeled as interacting entities. Kondor et al. [139] introduced *covariant compositional networks* (CCN) which modified the GNN architecture by extending the framework to incorporate symmetry and group theory. This allowed the network to handle more complex and structured data with inherent symmetries, as they often appear in physics. In [242], an encode-decoder architecture operating on graphs was proposed to simulate complex physical phenomena such as water flow, sand friction and viscous material. Li et al. [165] proposed two types of edge- and node-focused GNNs to accelerate Lagrangian fluid simulations. Cranmer et al. [60] used GNNs as an inductive bias for SR to learn the dynamics of spring-mass systems and dark matter halos for cosmology. The GNN architecture can be designed in a way that decomposes the problem into smaller subproblems of pairwise interactions be-

tween entities. These subproblems are easier to solve for SR algorithms and are a promising technique for high-dimensional problems, where conventional SR methods often fail. Further details on high-dimensional SR will be discussed in Sec. 3.2.5.

3.1.2 Non EC-Based Symbolic Regression Methods for Physics

ANNs typically operate as black boxes, offering little insight into the underlying physical laws they model. This lack of interpretability is a significant drawback for model inference in science and engineering, where understanding the governing principles is as crucial as making accurate predictions. Thus, the interest in SR methods for physics has grown, with approaches specifically designed to handle and reveal the underlying principles of particular physical systems. Subsequently, the most significant approaches outside the evolutionary computation domain will be presented.

SINDy A prominent technique for the identification of symbolic models for dynamical systems using prior knowledge is *sparse identification of nonlinear dynamics* (SINDy) [32]. It uses a set of selected basis functions that frequently appear in the governing equations of dynamical systems, such as trigonometric and polynomial functions. Sparse regression is applied to this set of functions to identify the terms actively contributing to the target variable and to generate parsimonious models. In this way, the property of many dynamical systems to consist of only a few relevant terms is exploited. Various publications have demonstrated the success of the method, even for long-standing problems in science [32, 64, 125]. However, its applicability is limited to identifying models that consist exclusively of the provided functional terms, which is a design decision influenced by domain knowledge. Moreover, the performance on high-dimensional problems can suffer due to the large number of candidate functions in the library, making the sparse regression step challenging. Overall, SINDy is best suited for relatively low-dimensional dynamical systems, which can be described by a sparse set of functions and for which prior knowledge about the function library is available.

AI Feynman Another physics-inspired method for symbolic regression is AI Feynman [272]. It aims at identifying functions of practical interest, which often share certain characteristics such as symmetries, separability, as well as consistency in terms of physical units. The algorithm recursively detects simplifying properties and divides the problem into smaller subproblems. In a first step, a dimensional analysis component takes the units of the variables into account and matches combinations of these variables with a given target unit. AI Feynman showed that unit information can be valuable to the algorithm, since many benchmark equations were identified correctly solely through dimensional analysis. However, this approach requires all units to be known in advance, including those of physical constants, which are often unavailable when searching for new empirical equations with unknown constants. Nevertheless, the dimensional analysis component can be considered a counter-movement to contemporary machine learning methods, which often standardizes features into dimensionless quantities. The Feynman dataset, containing 100 equations extracted from the *Feynman's Lectures on Physics* textbook, served as a benchmark for AI Feynman. The algorithm discovered all equations correctly, and achieved a 90% success rate on 20 more difficult equations.

An enhanced algorithm AI Feynman 2.0 was presented in [271], with an improved symmetry detection component using gradient information of a neural network fit, Pareto-optimality to evolve the most accurate model for each complexity level and better robustness to noise.

Neural Network-Based SR Petersen et al. [207] introduced *deep symbolic regression* (DSR) to formulate SR as a reinforcement learning task employing an *recurrent neural network* (RNN)-based architecture to construct symbolic trees. Mathematical expressions are treated as sequences of tokens, which are generated using probability distributions over the next token given the previous token. A risk-seeking pol-

icy gradient was utilized to train the RNN. This method showed competitive performance for simple physics problems [207] and the development of control policies [157]. Building upon this seminal work, Tenachi et al. [267] proposed DSR for physics, which is guided by unit constraints. Dimensional analysis was integrated as in situ constraints, which means that probability distributions for the next token are adjusted to filter for unit-conformal operations and features. This method also allowed the accommodation of other constraints, such as limiting the generated tree to a certain depth or length. A unified DSR framework was presented in [156], which leverages multiple previous concepts studied in the literature, such as recursive problem simplification as included in AI Feynman [272], population seeding [189] as well as linear models similar to SINDy [32].

The area of ANN-based SR is currently emerging rapidly, with many new approaches frequently presented that are of relevance for science and engineering problems. These include, but are not limited to, dynamic adjustment of the network architecture [164], learning ordinary [62] and partial differential equations [31], transformer-based architectures [124] as well as deep generative SR [114].

Other Approaches

A deterministic approach for developing symbolic models of practical relevance is *fast function extraction* (FFX) as presented in [172]. FFX uses pathwise regularized learning, which prunes a large set of candidate basis functions to form compact models. Reinbold et al. [220] applied sparse regression on high-dimensional experimental data. Their hybrid approach allowed the inclusion of problem-specific known physical constraints. The machine learning tool QLattice was inspired by Richard Feynman's path integral formulation [29]. A symbolic model is represented as a graph which is sampled from a probability distribution. This initially uniform distribution is adapted in an iterative process, which for faster convergence requires indication of preferences by the user. Cornelio et al. [55] proposed AI Descartes, which combines logical reasoning with symbolic regression to obtain symbolic models which are consistent with background theory. Domain knowledge, or background theory, is provided as general logical axioms which must be satisfied by the discovered symbolic models. Symbolic regression is performed through mixed-integer nonlinear programming.

3.2 Recent Advances in EC-Based Symbolic Regression

Physics-inspired SR approaches are sometimes limited to specific system types or predefined forms of equations, which can restrict their ability to discover new or unexpected physical laws and reduce their flexibility when applied to diverse or complex physical systems. GP can be considered a flexible approach for SR which allows for extensive customization, enabling it to evolve a wide variety of equation forms and adapt to different problem domains. This section presents recent developments in the GP domain relevant for the identification of symbolic models for science and engineering problems as outlined in the literature. The issues in GP identified by O'Neill [195] serve as a starting point for exploring the relevant aspects that require improvements. This list of issues was released shortly after the seminal paper by Schmidt and Lipson, which established the foundation for automated symbolic model discovery from experimental data in scientific research [246].

3.2.1 Generalization, Bloat and Overfitting

The Issue

A property of a good symbolic model is high generalization capabilities, i.e., not overfitting the training data [195]. The issue of overfitting is closely related to bloat, which describes the uncontrolled growth of an equation while leading to only small improvements in terms of error. It is also possible for overfitting

to occur in the absence of bloat, as related studies on bloat and overfitting suggested [68, 37]. Models that generalize well are of particular importance for scientific domains such as engineering or physics, as high-quality, comprehensive datasets may be limited, making generalization essential for robust insights.

Countermeasures Against Bloat and Overfitting

Various methods have been proposed in the literature to counteract overfitting and bloat towards models with better generalization capabilities. Smits et al. [258] proposed ParetoGP, which turns GP into a multi-objective problem by optimizing both an error measure and a complexity measure at the same time. Non-dominated solutions are added to an archive, which allows the user to select a good trade-off solution upon algorithm completion. In addition to the Pareto-dominance approach, the authors introduced an upper limit of equation growth to avoid exploration of unreasonably large solutions. The proposed algorithm sped up the evolutionary search and provided an effective countermeasure against bloat. In [87], Fitzgerald et al. suggested using bootstrapping to prevent overfitting. They sampled various bootstrap sets from the original training data with replacement and measured the standard deviation of the errors on the resampled bootstrap sets. This gave an indicator of the model sensitivity to small variations in the training data. While the performance on the test set was comparable to the one of standard GP without bootstrapping, the method prevented uncontrolled program growth. Mousavi et al. [188] aimed towards more generalizing models by controlling their first-order derivative. In their study, the *rooted mean squared error* (RMSE) between the numerically calculated true derivative and the derivative of the model was considered as a proxy for model complexity and was optimized using a multi-objective algorithm. The underlying assumption was that simpler derivatives implied less intricate behavior, while complex models often have erratic derivatives, which can imply overfitting. Experiments confirmed this assumption and the efficacy of the method. In [48], Chen et al. proposed angle-driven geometric semantic operators for *geometric semantic genetic programming* (GSGP), which frequently faced problems with overfitting and bloat. Their approach generated smaller models and improved the generalization performance.

Measures for Model Complexity

Measuring bloat is highly related to defining appropriate measures for the complexity of a symbolic model. Typical complexity measures are the number of nodes or the depth of a GP tree. Vanneschi et al. [276] proposed a simplified calculation of the function curvature as an estimate for the function complexity. Furthermore, they interpreted bloat as the relationship between the average length growth and the average fitness improvement up to a certain generation compared to the respective values at generation zero, where no bloat occurs. Generalization was measured using a validation set next to the training data. In [258], the sum of the number of nodes of a tree structure and all its subtrees was used as a complexity measure. This setup furthermore preferred balanced trees with lower maximum depth over more complex tree structures. Vladislavleva et al. [283] used the same complexity measure as [258], and furthermore proposed the order of nonlinearity as an additional measure to discourage highly nonlinear and volatile responses to tiny changes in the input. The order of nonlinearity was defined as the minimum degree of a Chebyshev approximation polynomial of a function. While the approach produced compact models that generalized well, a major downside of this method was the expensive approximation of the Chebyshev polynomial. Kommenda et al. [136] proposed a recursive complexity computation procedure that incorporates the semantic information of operations. Functions such as a square or trigonometric function increased the complexity value more than addition or subtraction operations, which are generally considered simpler [88]. Chen et al. researched on sophisticated complexity measures to enhance generalization of GP for SR, such as the Vapnik–Chervonenkis dimension [52] and Rademacher complexity [49]. While the former had the drawback of high computational cost to compute the Vapnik–Chervonenkis dimension, Rademacher complexity was beneficial for model generalization capabilities and interpretability.

Three doctoral theses from renowned researches in the area of GP focused on the issues of bloat, generalization and overfitting in the past. Kronberger in his thesis [145] addressed different methods to reduce the model complexity. Kommenda [134] introduced the recursive complexity and performed several case studies on noise-free and noisy datasets, as well as adaptations of the *non-dominated sorting algorithm* (NSGA)-II algorithm to control complexity. Chen in her thesis [44] focused on generalization of GP for SR and improved generalization of models obtained with GSGP.

As it became apparent from the literature reviewed in this section, there exist various ways to measure complexity, and different algorithms to improve generalization and avoid bloat and overfitting. There is no one best way to measure complexity, but rather this is a personal choice depending on the specific application. The approach of simultaneous multi-objective optimization of an error and a complexity measure is a widely accepted method to tackle these issues in GP.

3.2.2 Constants in Genetic Programming

The Issue

The identification of numerical constants, i.e., their position and value within a symbolic model, is a challenging task, due to the infinite space of possible real values. At the same time, constants are of high importance in regression models, as they help to fine-tune models to fit data accurately and also play a role in the model interpretation. For a long time, *ephemeral random constant*s (ERCs) were the standard approach for introducing new constants to symbolic models in GP. The value of an ERC is randomly generated during the initialization of a tree and remains constant within that tree unless changed by a mutation operator [210]. As it is unlikely that the initially random value of an ERC minimizes the overall error, various approaches have been proposed to modify them [239]. Topchy et al. [269] introduced a constant tuning algorithm based on gradient descent in 2001, leading to improvements in terms of final fitness and reduced time to reach this final fitness. However, as also Kommenda pointed out [134], this method did not make its way into standard GP implementations until a few years ago, which might have been due to the increased number of hyperparameters that needed to be defined for the gradient-based constant tuning algorithm.

Overall, two major streams for constant learning in the literature can be identified: avoiding the fitting of constants inside a GP tree, and fitting constants in a tree using a gradient-based algorithm. Several publications used different wordings for constant learning, such as parameter identification, parameter fitting or coefficient estimation, which can be employed interchangeably.

Avoiding Constant Learning Inside GP

Various methods are available in the literature where no internal constants in the GP tree were learned. Keijzer [127] improved GP with linear scaling, where model outputs were adjusted using a linear transformation. The idea is that GP can focus on learning the right shape of the equation, which is wrapped into a linear scaling algorithm that learns the slope and intercept to scale the output values to the target variable. In comparison to the gradient-descent based constant optimization proposed in [269], the linear scaling approach achieved lower error rates. However, this method limits the searched models to linear ones, and cannot learn arbitrary nonlinear correlations. Multigene GP [253] can be considered an extension to linear scaling, which involves linear combinations of non-linear transformations of the input variables. Rather than wrapping a single individual in a linear function, multigene GP evolves individuals that consist of multiple terms that are each scaled with a constant. Multiple regression GP [10] inserts constants at different positions within an individual and optimizes them using the least angle regression algorithm [77]. Instead of letting the GP algorithm find the optimal constant placement, specific rules are used to position constants within the tree. For the RMSE computation, the model coefficients need to be fitted for every individual before error calculation. Recently, Chen et al. [45] proposed a method which replaces the frequently used

RMSE as an objective with a correlation measure between model output and target variable. Using a correlation measure, no coefficients were fitted during training, and only a final linear scaling step was applied once the search was finished.

When coefficient fitting is applied during evolutions, GP algorithms must focus significant computational resources on this process, rather than searching for suitable model structures. Chen et al. [45] proposed to relieve GP from the task of positioning and fitting constants during evolution, and employed a correlation-based fitness function. This saved computation time and avoided the sometimes cumbersome optimization of ERCs. However, their linear scaling approach limited models to linear combinations of terms and could not learn arbitrary nonlinear correlations, which is often required in empirical equation discovery.

Combining GP and Gradient-Based Constant Optimization

Next to the previously introduced methods, various publications on combining GP with an additional gradient-based optimizer for constants exist in the literature. This allows for more flexible positioning of constants within a tree, and avoids restrictions on the shape of the final solution.

Gradient-based algorithms are most widely used to optimize constants on the fly during the equation search of GP. While already introduced in 2001 [269], gradient-based methods took some time to make their way into standard GP frameworks. Chen et al. [47] investigated the performance of a hybrid GP algorithm with gradient-based constant optimization using different algorithm configurations. They varied over the number of gradient descent steps, number of individuals which received constant fitting, and time points within the evolution at which constant fitting was applied. While the overall performance improved, they pointed out that optimized constants could lead to overfitting on a test set.

Building upon these approaches, the work of Kommenda et al. [138] was considered path-breaking towards the inclusion of gradient-based constant optimization in GP libraries. They proposed a local search mechanism named *constant optimization with nonlinear least squares* (CO-NLS) [135, 138], which combines linear scaling and gradient-based optimization. CO-NLS constitutes two steps: first, the gradient is computed with automatic differentiation; second, the Levenberg-Marquardt algorithm is applied, which is specifically tailored to models that are nonlinear with respect to their parameters (see Sec. 2.1.2). Constants can appear at any leaf position within the tree, and the model output is additionally scaled by inserting slope and intercept coefficients. The beneficial effect of constant optimization was demonstrated on the same benchmark problems that had previously been used to compare linear scaling and gradient-based constant optimization in [127]. CO-NLS achieved error values of zero or close to zero on these problems, and the efficacy of the approach was demonstrated in further studies [134, 137].

Despite the large benefit in terms of model error, this approach had two major drawbacks: first, the computational overhead attributed to gradient computation and optimization; and second, the dependency on the starting values, which determined whether the algorithm converged to a local or a global optimum. The latter could be solved by repeating the constant fitting process multiple times with different starting values, which in turn increased the computational effort. Overall, the advantages seemed to outweigh these issues, so that multiple software implementations of GP have since made a gradient-based constant optimizer the standard, such as Operon [34] and `TiSR` [170], which both employed the Levenberg-Marquardt algorithm. Cranmer et al. [57] in the `PySR` framework used the *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) algorithm as a non-linear constant optimizer. Research on constant optimization in GP represents a significant area of interest within the academic community, as recent studies on feature standardization and coefficient optimization [70, 230], linear scaling [192, 45] and applications to real-world problems [259] demonstrated. A recently proposed multiview GP approach [236] enables the learning of struc-

turally equal expression with different coefficient values. This can be useful for various problems that stem from the same problem family, but have different experimental hyperparameters.

3.2.3 Distributed Genetic Programming

The Issue

An appealing characteristic of GP, or generally *evolutionary algorithm* (EA), is the possibility of parallelization across multiple processors or computing nodes [e.g., 266]. A prominent method to this end is the so-called *island model genetic programming* (IMGP) [7], which divides the population of individuals into multiple subpopulations, which are evolved in parallel on different "islands". In this way, the workload is divided, and the computation sped up, which allows for scalability in terms of larger datasets or more intricate problems. IMGP requires the definition of multiple additional parameters, including the number of islands on which the population is distributed, the migration topology that specifies migration pathways, and the migration rate that balances information transfer between islands to increase diversity while allowing enough time for optimization. Additionally, the number of individuals migrating at each phase needs to be specified. Next to the obvious advantage of improved speed, IMGP also facilitates the exploration of different areas of the search space and increases diversity within the subpopulations through migration. This approach helps to prevent the algorithm from premature convergence, and can result in increased search efficiency and probability of success in finding a good, or, in case of known benchmark equations, the correct solution. O'Neill et al. pointed out that the identification of an appropriate setting of these hyperparameters is an open question in GP research that deserves attention [195]. Two different research directions contributing to the improvement of IMGP were identified: First, relatively general IMGP approaches focusing on the identification of appropriate migration topologies and optimizing their parameters. Second, improving upon the basic IMGP by defining an island and migration scheme that is based on certain characteristics of an individual, such as fitness, age, or frequency of subtrees.

General IMGP Approaches

Generally, the migration topologies are not GP-specific but can be borrowed from the broader EC area. Different migration topologies have been proposed and assessed in the literature [82, 212, 83]. The most prominent ones were the ring, the grid, the mesh and the random topology. While ring, grid, and mesh topology define a specific migration pathways between the islands, the random approach migrates individuals from a randomly chosen island to another. Fernandez et al. studied the influence of population size and number of populations [81], as well as migration topologies and migration rates [82]. Comparing the influence of the parametrization on three test problems, the authors observed negligible differences between different migration topologies, with the relatively simple and least predetermined random topology performing at least as good as the others. Larger effects were apparent for the frequency and quantity of migrated individuals, where the authors recommended migrating few individuals more frequently instead of exchanging larger amounts of individuals with larger intervals.

While the previously introduced papers focused more on the population dynamics, Fillon et al. first addressed the issue of maximizing the success rate with IMGP [86]. Their *divide and conquer* (D&C) strategy used different function sets for each island, and outperformed the single population algorithm, even when the latter had a larger population size. However, the distribution of function sets across different islands required highly context-specific knowledge, introducing an additional design decision that had to be made in advance.

Sophisticated Migration Strategies

Different approaches aimed to enhance the general IMGP setup through new migration strategies exist in the literature. Hu et al. [118] proposed a *hierarchical fair competition* (HFC) model for parallelized evolution. HFC was inspired by a stratified competition as often encountered in society or biology, where individuals only compete with others that have similar skills. Rather than

a random initial assignment of individuals to an island, the population was divided into hierarchically structured subpopulations with increasing fitness values. Individuals with lower fitness values started off in the lower subpopulations, and were admitted to a higher subpopulation once they passed the (higher) admission threshold of that layer. To maintain diversity and prevent the algorithm from premature convergence, new randomly generated individuals were frequently introduced in the lowest subpopulation. Using fitness to segregate individuals can, however, cause unexpected difficulties. When individuals converge to a local optimum at the top of a fitness layer, they can block new individuals in different regions of interest from advancing, as pointed out in [115]. A modified version of HFC with adaptive fitness thresholds [119] was proposed to overcome this issue, but did now show significant improvements to the original.

Hornby proposed the *age-layered population structure* (ALPS) to mitigate premature convergence in evolutionary algorithms [115]. Rather than relying on the fitness values as a segregating factor between layers or subpopulations, the age of an individual determined the subpopulation to which it was assigned. Age here referred to the number of generations in which the genetic material of an individual had evolved. Thus, the offspring received the age values of its oldest parents plus one. In this way, a structure similar to school grades where students start in lower grades and proceed to higher grades as they age was created. To maintain diversity, new genetic material in the form of randomly generated solutions was introduced in the lowest layer, similar to the HFC approach. ALPS significantly outperformed HFC in the original implementation on different test problems. An extension to ALPS, the so-called steady-state ALPS for real-valued problems, showed superior performance on multi-modal problems compared to differential evolution and evolution strategies [116].

Ono et al. [199] proposed a migration scheme based on frequent subtrees which appear in a subpopulation. Making use of the building block hypothesis, they assumed that subtrees that appeared frequently among the individuals of an island also contained valuable genetic material for the later generations. An activation measure was introduced that considered not only the fitness of an island, but also how many types of frequent subtrees had been generated. To detect and compare subtrees reliably, a labeled ordered tree representation was used. Individuals from islands with high activation levels were migrated to islands with low activation levels to promote the spread of beneficial building blocks, improving overall convergence and performance. The SR benchmark equation in that paper was approached in a single-objective manner using the approximation error as the objective. In this setting, the proposed method outperformed a standard IMGP implementation on three benchmark problems, and furthermore introduced more new subtrees in the population, while the standard IMGP introduced no new subtrees after some generations. An extension of this approach [198] using an additional local search mechanism furthermore outperformed ALPS and standard IMGP on six benchmark problems. However, on the SR problems, the differences between these three methods were smaller than on other problems, and all three performed significantly better than a GP algorithm without island models. Furthermore, the quality estimate was mainly based on the final fitness values, and no information on the shape or complexity of the solutions was given.

One Setting Fits All?

Overall, both research on the general and on the enhanced migration IMGP methods indicated that it is very difficult to identify an overarching parameter setting that performed well on different kinds of problems [198]. Recently, Burlacu et al. [35] compared a canonical GP algorithm, ALPS, and other methods for population diversification in GP using five benchmark problems for symbolic regression. The authors observed different behaviors between the tested algorithms, and also between different problems solved by the same algorithm. They concluded that possibly there might be no optimal algorithm parametrization for equation search, as the behavior also depends on the spe-

cific problem characteristics. Furthermore, the enhanced IMGP approaches, which incorporated sophisticated migration strategies, required the specification of additional parameters, involved higher implementation effort, and, in cases where frequent subtrees were controlled, relied on a specific tree representation for efficient subtree counting. The complexity and parameter sensitivity of these enhanced migration methods made them challenging to apply universally in practice, as they required customization to the unique characteristics of each problem, limiting their use as a standard approach.

It appears that the relatively simple IMGP approach with an empirically selected set of parameters is the preferable option to the canonical GP without island model. Recently, IMGP with a modification of the random migration topology was implemented in a state-of-the-art GP framework, `PySR` [57]. This framework uses an archive of the best solutions among all combined subpopulations. Random individuals from a subpopulation are exchanged by randomly selected solutions from the archive. A similar approach is used in the `TiSR` framework [170].

Open questions in this area remain. The IMGP methods introduced in this section mainly used single-objective optimization on symbolic regression problems [86, 198]. As typically multiple objectives are considered in contemporary GP, further research is necessary to evaluate how IMGP and multi-objective optimization influence each other with respect to the success rate.

3.2.4 Further Algorithm Components to Improve GP

The Issue

Another issue highlights the complexity of a GP system and the arising difficulty in understanding and predicting how various components of a GP algorithm interact. Various methods to handle each stage of a GP algorithm have been presented in the literature, providing different strategies for population initialization, selection and replacement, as well as genotype-phenotype mappings. However, it is nearly impossible to predict the behavior of such as system of algorithm components and choose optimal configurations accordingly [195]. In the last decade, further variations of algorithmic components have been presented in the literature to enhance GP for SR tasks. In the following, an overview of interesting approaches from the literature is given.

Model Improvement and Regularization Techniques

Various publications focused on methods to enhance the performance and robustness of symbolic models developed by GP. Keijzer [127] proposed the use of interval arithmetic and linear scaling to optimize model precision and adjust for better performance. Interval arithmetic is a method used to estimate the upper and lower bounds of an arithmetic operation based on the input values of the variables. In this particular case, it was used to circumvent a combination of functions and input variables that resulted in undefined values. Experiments on a diverse set of benchmark functions indicated the superior performance of linear scaling compared to gradient-descent GP and unscaled GP. Although introduced in 2003, this method remains a valuable tool for GP improvement to this day. Interval arithmetic is a popular method to introduce shape constraints in GP [e.g., 104, 106], and linear scaling is a well-established method to improve GP and relieve it from constant fitting [192, 45].

Kinzett et al. [133] investigated how two online simplification strategies, namely algebraic and numerical simplification, affect the search for GP models. Experiments on classification tasks indicated that methods are effective in generating new diverse building blocks during evolution while retaining many existing ones, although they may also eliminate some. Compared to the canonical GP, these simplification methods produced smaller programs, require shorter evolutionary training times, and achieve similar effectiveness. Online simplification was also implemented in popular GP frameworks that have recently been published [57, 170].

Schmidt and Lipson [248] offered an alternative perspective on noise in data for SR problems. The conventional assumption was that noise tends to average

out and is nearly symmetric, which led to the belief that MSE or maximum likelihood estimators were suitable loss functions for the algorithm. In contrast, they introduced a stochastic element as a novel variable added to the terminal set, designed to model noise in experimental data throughout the GP tree. Unlike other variables, this terminal value is randomized at each evaluation, even if it appears multiple times within the same equation.

In a later publication, Schmidt and Lipson [245] present *age-fitness Pareto optimization* (AFP), which is inspired by the previously introduced ALPS [115]. Instead of dividing the population into separate layers based on age, the age of an individual was included as an additional objective tackled with multi-objective optimization. In this way, young individuals remained in the population as they were non-dominated on the age objective. Results strongly suggested that the multi-objective approach performed better on various benchmark problems compared to ALPS and the canonical GP algorithm.

Virgolin et al. [281] proposed a new approach to evolve interpretable and accurate equations with GP in a multi-objective manner. Typically, an error measure is optimized alongside a complexity measure, and less complex equations are assumed to be more interpretable according to Occam's razor. In their algorithm, the complexity measure was replaced by a formula of interpretability, which was learned from the results of a survey on interpretability among scientists who frequently worked with mathematical expressions. This formula considered the overall length of an equation, but gave more weight to the number of non-arithmetic operations and the number of consecutive non-arithmetic operations, as they deteriorated interpretability. Compared to the typical bi-objective error and complexity optimization, their methods reached more concise and interpretable expressions with similar or higher accuracy.

Owen et al. [201] introduced an extended bias-variance decomposition that separated the model error into bias, external variance (due to limited sampling), and internal variance (from algorithmic randomness). Applying this method to GP revealed how parameters such as the tree depth, the number of generations, the function set complexity, and data standardization affected the predictive performance. This approach enabled more insights into the influence of each GP component on the model accuracy and helped to select suitable algorithm components.

The same researchers investigated feature standardization for GP [70, 202], where a Z-score standardization of both inputs and responses ensured that evolution operated within a data space with zero mean and unit variance. Analysis of several benchmark datasets suggested an algorithm performance comparable to advanced symbolic regression methods, while generating simple yet effective symbolic models. Furthermore, feature standardization enhanced coefficient optimization through gradient descent for accurate model production.

Hybrid and Advanced Algorithmic Techniques

Next to the methods to improve model performance and robustness, various hybrid and cutting-edge approaches that combine multiple methods or introduce new algorithms to advance genetic programming have been developed lately.

La Cava et al. [151, 154] presented an ϵ-lexicase selection approach as an extension of the original lexicase selection approach [260], which had previously performed poorly on continuous SR problems. Lexicase selection follows the idea of rewarding individuals with outstanding training performance, by evaluating candidates based on a sequence of test cases, selecting those that perform well on any given case. ϵ-lexicase selection is an improved method for continuous problems which introduces a tolerance threshold, allowing for more flexible candidate selection by accepting solutions within a small error margin. This method preserved diversity within the population and outperformed other diversity-maintaining methods from the literature.

The *feature engineering automation tool* (FEAT) proposed by La Cava et al. [153] represented features using syntax trees that resembled neural network architectures. Optimization of these structures was achieved through the use of

a multi-objective evolutionary algorithm, which optimized an error, the recursive complexity as proposed by Kommenda et al. [134] and an "entanglement" objective. Disentanglement refers to a representation's ability to separate factors of variation in the underlying process. Minimizing entanglement involved reducing connectivity within the network to improve interpretability. FEAT demonstrated superior performance across a set of one hundred problems, yielding representations that were by several orders of magnitude smaller than those produced by the runner-up method.

Sun et al. presented a novel memetic evolutionary algorithm using a fractional representation for SR [264]. A memetic algorithm is an optimization technique that combines evolutionary algorithms with local search methods to refine existing solutions. The proposed approach used analytic continued fraction representations to derive rational function models. The algorithm included a population of computational agents that improved solutions and searched for models with fewer variables, constrained by a tree-based population structure to enhance consistency and performance.

De Franca et al. [93] introduced an *interaction transformation evolutionary algorithm* (ITEA) for SR as an extension of the original algorithm [4, 92]. ITEA is a mutation-based algorithm that employs an interaction-transformation representation. This representation linearly combines small nonlinear terms with respect to the original variable, thereby restricting the search space to simple expressions. The interaction strength is defined by the exponent of a variable and indicates the degree of interaction between multiple variables within the model, which is adapted during the search for symbolic models. In this way, concise models with competitive or superior accuracy compared to other state-of-the-art regression methods were evolved.

Further developments in this area include a Zoetrope GP approach for regression [25], which introduced a dynamic representation in genetic programming that uses repeated fusion operations between partial expressions. The newly evolved features are combined linearly to create an individual, and new individuals are formed using representation-specific variation operators inspired by geometric semantic crossover [187]. The *gene-pool optimal mixing evolutionary algorithm* (GOMEA) [280] was designed to evolve small, interpretable expressions for symbolic regression tasks. In each generation, a model-learning phase is included, where a statistical model of interdependencies within parts of the genotype is developed. This model was used to propagate genotype patterns and preserve their joint influence. Mundhenk et al. [189] introduced neural-guided population seeding, which is a hybrid approach combining RNNs with population-based GP. An RNN is trained on intermediate models of the GP population, and the GP search is restarted multiple times with increasingly improved starting populations sampled from the RNN. Taylor GP [108] employed Taylor polynomials to extract features such as polynomial order, variable separability, and monotonicity to decompose the problem.

Geometric Semantic Genetic Programming

Another notable research direction to improve GP for SR is GSGP [187, 273, 275]. In GSGP, the typical crossover and mutation operations are replaced by special ones that act directly on the semantics of a program. For regression tasks, semantics refers to the vector of output values, which can be represented as a point in the n-dimensional semantic space when the number of output values is equal to n. An interesting characteristic of using geometric semantic operators for crossover and mutation is that they produce an unimodal error surface for any supervised learning problem where fitness is calculated using an error measure between outputs and targets. This error surface on the training data has a global optimum, which renders this approach especially promising for complex optimization problems containing big amounts of data [38]. Pawlak et al. [205, 206] introduced a set of geometric operators designed for population initialization, mate selection, mutation, and crossover for symbolic regression. The operators were theoretically justified and experimentally compared to traditional methods for SR and function synthesis, demonstrating

superior performance in best-of-run fitness, test-set fitness, and program size.

Recent improvements on GSGP include the combination with linear scaling by Nadizar et al. [192], which improved the performance compared to the standard GSGP on five synthetic and six real-life benchmark datasets. However, their observation raised the concern of overfitting for some cases when GP or GSGP was combined with linear scaling. Furthermore, Vanneschi addressed the frequently appearing issue of bloat in GSGP and introduced SLIM_GSGP [274]. A novel deflate mutation operator created smaller offspring from a parent individual, which was effective in creating more concise yet equally good solutions compared to traditional GP and GSGP methods.

After presenting and analyzing approaches from the literature that aimed to enhance existing GP components or proposed new ones, it became apparent that selecting the most suitable algorithm components remains an open issue in GP. While various iterative improvements of specific algorithms could be observed over time, a single optimal design could not be identified. Competitions to compare SR algorithms on benchmark suits revealed which algorithms may perform better than other on problems with specific characteristics. A number of comprehensive review papers on contemporary SR methods, as well as the analysis of tournaments to compare their performances on a diverse set of benchmark problems, include [91, 152, 169, 200, 213].

3.2.5 High-Dimensional Genetic Programming for Symbolic Regression

The Issue

O'Neill et al. raised the issue of scalability in GP in their position paper in 2010 [195], which was discussed in two separate but connected ways. Here, we refer to scalability being defined as "the ability to provide algorithmic solutions to problems that are of substantial size/dimensionality" [195]. The authors highlighted the importance of modularity in GP to tackle these issues. Modularity refers to the ability to break down a complex problem into simpler, reusable subcomponents. Early works on introducing modularity in GP included the evolution of *automatically defined functionss* (ADFs), which allowed the creation of modular code, where certain parts of a solution was reused across different parts of the main program [122, 141].

High-dimensional datasets are common in the science and engineering fields, often leading to an explosion of the search space due to the numerous features and variables, which complicates the search process. Issues such as overfitting and the evolution of overly complex models appear frequently, making this topic closely related and often addressed alongside the issue of generalization [49, 296]. Different definition of "high-dimensionality" exist in the literature, for instance datasets with more than 50 variables [296] or datasets with more variables than samples [110].

While high-dimensional datasets are often encountered in GP, methods to tackle these problems have mainly focused on high-dimensional classification problems. High-dimensional SR tasks have primarily gained importance in the last few years [50, 110]. Mapping the idea of ADFs to symbolic regression, a potential solution could be the construction of features from the original (high-dimensional) variable space which can be reused in an expression. Next to feature construction or extraction, other methods in the literature to tackle high-dimensional problems involve feature selection and a physics-inspired inductive bias for modularity. In the following, an overview of the relevant contributions in the literature will be given.

Feature Selection

In many high-dimensional datasets, variables are often correlated or have a minimal effect on the target variables. Consequently, identifying and selecting a subset of significant variables becomes crucial. Feature selection aims to reduce the dimensionality of the dataset by including only the most impactful variables. GP inherently includes a feature selection mechanism, which reaches its limit when the number of variables surpasses a certain threshold. Chen et al. [46] presented one of the earliest works on GP for high-dimensional sym-

bolic regression, which embedded a feature selection phase in the overall GP algorithm. In a first stage of their proposed algorithm, important features were selected from the fittest individuals, which were used as the input features in the second stage where GP was applied on the reduced set of variables. A different approach by Chen et al. [50] investigated feature selection with permutation importance. A subset of features was selected in a GP-based preprocessing step, which served as an input to the training of the final regressor using GP. Dick et al. [69] employed a similar variable importance measure that involved permuting variables in an expression. They assumed that the shuffled variables causing the highest change in the fitness function on a holdout dataset were the most important.

Zhong et al. [298] compared multiple methods to reduce the numbers of features of high-dimensional SR datasets before applying GP. Experiments on five datasets, with the number of variables ranging between 53 and 280, suggested that the principal component analysis, which projects the data into a lower dimensional space without retaining the original features, performed the worst among the tested methods. Their proposed feature selection method based on the maximal information coefficient performed best when both correlation and redundancy were considered. Arslan et al. [11] proposed a method based on artificial bee colony programming.

In [231], a method performing feature selection using a cooperative approach employing an EA and GP was proposed. In the EA, features were represented as binary strings, where a value of 1 indicated that the feature was selected. The quality of an individual in the EA was determined by the best fitness from an entire run of GP. While this approach was extremely time-consuming due to the interplay between EA and GP, it demonstrated comparable or better performance on various classification tasks.

Recently, Al-Helali et al.[110] presented the *remGP* method, which evaluated feature importance by assessing the impact of their removal. Unlike previous methods that shuffled features to measure their importance, as done in [50, 69], *remGP* systematically removed features from an evolved expression to observe their effect on the final result. This approach did not rely on precomputed statistical properties to remove irrelevant features on a data level, but allowed for feature selection to be performed directly on the evolved expressions.

Zeng et al. [296] introduced a gradient-based approach to optimize the structure of a GP tree, which used a novel differentiable symbolic tree representation. The typically discrete function and terminal sets of GP were relaxed into a continuous representation, which allowed for adaptations of the tree structure through gradient descent. In this continuous space, a GP tree was defined solely by a node matrix and an adjacency matrix with learnable weights. While sticking to the general idea of evolving a population of trees, trees were converted into the continuous representation and the training process boils down to optimizing the two matrices. Optimization phases took turns with diversification phases, where operations similar to mutation and crossover were performed on the trees. This method achieved notable results on SR benchmark datasets with up to 7400 variables, and outperformed canonical GP [90] and also more advanced GP approaches such as the previously described Rademacher complexity for improved generalization [49] and bi-objective GP for interpretability [281]. Interestingly, all GP-based approaches achieved better results than the hybrid neural-guided population seeding [189] and deep symbolic regression [207] (further details in Sec. 3.1.2).

Feature Construction and Extraction

The previous methods made high-dimensional problems more manageable for GP by removing correlated or redundant features from the dataset, thereby transforming the problem into a lower-dimensional one with fewer variables. However, in some cases, even a well-designed model may require most or all of the variables [173]. This is a scenario which is especially likely in scientific problems and pertains to a long-standing challenge in data-driven ML [208]. In such cases, feature selection may not be suitable, as it could remove impor-

tant information when each variable contributes to different dimensions of the prediction. Feature construction or extraction aims at identifying meaningful features from the original variables, rather than removing seemingly irrelevant information. To this end, various methods for introducing modularity to facilitate the decomposition of the problem into more manageable units for GP have been presented in the literature.

One line of research involves the dynamic extraction of these features during the evolutionary process. In [173], an approach based on latent variables and nonlinear sensitivity analysis was presented to identify mathematical expression from high-dimensional datasets. The approach was tailored to cases where almost all variables were required to capture the variation in the data with respect to non-linear sensitivity analysis between input and target variables. The method made use of the concept of *latent variable symbolic regression* (LVSR), where each latent variable t is a linear combination of the input variables, resulting in a one-dimensional variable. Different methods to adjust the parameters in the linear mapping were assessed, such as random forest and linear regression using lasso and ridge regularization. GP learned a nonlinear function of the latent variable $g(t)$) to fit the training data. More latent variables and nonlinear functions of these were added iteratively to the model until a maximum rank or a desired accuracy was reached. Comparing the LVSR performance with classical regression methods and a GP algorithm with feature pruning on a dataset with 240 variables demonstrated the effectiveness of this approach.

One of the few papers explicitly targeting feature construction for high-dimensional SR tasks is [51], where Chen et al. proposed a method based on frequently appearing building blocks. In a first step, individuals with higher fitness gains compared to their parents were analyzed in terms of depth and activeness of a building block. While depth referred to the number of levels in a building block, activeness counted the frequency of a building block within the analyzed individuals. Building blocks within a predefined depth range and activeness level were transformed into new features. In the second step, these features were introduced into the set of terminal within the GP algorithm. Furthermore, some individuals with high error values were removed from the population, which was seeded with new individuals that used the precomputed building blocks. Compared to standard GP, the proposed method evolved more compact models with better generalization capabilities, as experiments on SR datasets with 122 to 7399 features demonstrated.

Further examples for feature construction for high-dimensional classification problems were presented in [240, 270]. An interesting approach is *cooperative coevolutionary genetic programming* (CCGP), which demonstrated competitive performance on a high-dimensional classification task [232]. However, as recent research by Zille et al. indicated, CCGP did not necessarily produce better results than canonical GP for regression tasks [299].

Another noteworthy line of research exploits specific problem characteristics to divide high-dimensional problems into smaller chunks and make them thereby tractable for GP. In this way, the search space and computational complexity can be reduced, and the performance and efficiency of GP can be enhanced by focusing the search on relevant components of the problem. This typically requires some knowledge about the problem or the domain, which is often available for applications in science, engineering, or physics.

Lou et al. [168] utilized the separability inherent in many problems in science and engineering and proposed a D&C approach to tackle such problems. The target function was divided into a number of subfunctions by the novel bi-correlation test. These sub-functions were then each approximated separately with a GP algorithm. Experiments indicated an accelerated convergence speed of GP, since only smaller components and not the interactions of all variables had to be learned. However, although this approach was developed to tackle high-dimensional problems, the two benchmark datasets contained only two

and five variables. While this approach was previously limited to models with additive and multiplicative separability, Chen et al. [43] overcame this issue by introducing a bi-level D&C approach with an additional factoring component. Their *block building programming* (BBP) approach divides the separability detection into a first block and a second factor detection step. The algorithm outperformed a canonical GP algorithm in terms of convergence speed and accuracy for a problem related to the flow pass of a circular cylinder. Another extension of this method that allowed for the inclusion of the same variable in multiple function blocks demonstrated competitive performance on a problem [42]. However, the number of variables considered in the benchmark problem was again only small.

Modularity Through Physics-Inspired Inductive Bias

Cranmer et al. [60] presented an interesting hybrid method that employs GNNs with a separable internal structure as an inductive bias for GP and decomposes the problem into smaller sub-problems. The terms GNN and *message passing neural network* (MPNN) as introduced in Sec. 2.2.2 were used interchangeably here. Their approach leverages the ability of deep learning to operate on high-dimensional data, and symbolic regression to generate interpretable models. They proposed a framework for problems that can be modeled as interacting entities, as they appear frequently in problems studied in science and engineering. Since MPNNs can represent this underlying structure, an MPNNs was first trained on the available data, which induced a strong bias on the underlying interactions between entities. Compact internal representations were encouraged during training to reduce the dimensionality of the hidden vectors, making the symbolic regression more tractable. The internal parts of the MPNN, namely an edge and a node model which used shared parameters for each edge or node update, were then replaced by symbolic models. All incoming edge messages were summed into a function $f(x)$, and served as input to the node model $g(x)$, so that the final model was a nested function $g(f(x))$. In this way, the complexity of the problem for the GP (or any other SR) algorithm was reduced, and the number of variables the algorithm operated on was significantly reduced compared to the original dataset. The effectiveness of this approach was demonstrated by the successful rediscovery of Newtonian and Hamiltonian dynamics, as well as the improvement upon an existing expression for a problem in cosmology. This method furthermore rediscovered the gravity equation for international trade in [279], the collective behavior of swarms [211], as well as the orbital mechanics of our solar system according to Newton's law of gravitation [162]. While this approach demonstrated impressive capabilities to rediscover systems with known ground truth, further research is required to assess its performance on problems for which no underlying equation has been identified yet.

3.2.6 Genetic Programming Software Implementations

The Issue

The preceding sections illustrated the versatility of GP research and the recent advances. For each issue addressed, various research directions and contributions to the literature have emerged. However, this multiplicity of techniques and associated parameters and the challenge to determine a single best approach make it difficult to configure a GP algorithm for a given problem. O'Neill et al. [195] have already raised the issue of usability in 2010, and proposed to enhance it with the development of an easier to use and more intuitive software implementation, similar to a "one button" GP. Meanwhile, multiple easy to use and customizable GP frameworks with different characteristics have evolved. The most important ones will be briefly described here.

DEAP

`DEAP` [90] implements the canonical GP algorithm in Python. It supports a wide range of evolutionary algorithms and provides various customization options and components, which can be assembled into an algorithm using a toolbox. It is therefore often used to implement and prototype new algorithms, or as a baseline method when novel algorithms are compared to a GP baseline, for example for high-dimensional GP [296] and feature selection [110]. It uses

ERCs and its associated mutation operators rather than a separate nonlinear regression algorithm for parameter fitting.

GPTIPS `GPTIPS` [253] is a MATLAB-based platform for multigene GP, where a model is a linear combination of subtrees and coefficients fitted using a nonlinear least squares solver.

Operon `Operon` [34] is a framework written in C++ intended to achieve an efficient implementation in terms of runtime and memory, and offer a pre-configured algorithm where a user only need to add the corresponding dataset. It incorporates non-linear least squares constant optimization using the Levenberg-Marquardt algorithm.

Genetic Engine `Genetic Engine` is a hybrid between strongly-typed and grammar-guided genetic programming and uses a context-free grammar to guide the evolution of symbolic regression models. ensuring syntactic correctness and incorporating domain-specific constraints [80]. It allows for single- and multi-objective optimization, to provide domain knowledge about the shape of the solution using type annotations and the definition of problem-specific fitness functions.

Bingo The `Bingo` framework [216] evolves general acyclic graphs with a linear representation. The modular code structure allows for simple abstraction and exchangeable of existing or implementation of user-defined components. It furthermore employs a constant tuning algorithm, algebraic simplification, and coevolution of fitness predictors. The fitness function in Bingo is customizable and supports the computation of partial derivatives, which is useful to consider physics-related constraints in the fitness evaluation.

PySR A tree-based framework developed for practical application of SR in the sciences is `PySR` [57]. It constitutes a multi-population evolutionary algorithm, which executes an evolve-simplify-optimize loop. It aims at balancing accuracy and simplicity of a symbolic model, and uses a tournament selection-based reproduction method, where accuracy and simplicity are condensed into a single performance measure. A simplification phase helps to remove redundant operations in an individual. The BFGS algorithm is adopted for constant tuning, which is performed every couple of generations. Customizability is a key characteristic which allows for user-defined operators, custom fitness functions and further ways to influence the search such as weighting data samples or restricting the complexity and nesting of some operators. The algorithm maintains and returns an archive of Pareto-optimal solutions in terms of accuracy and complexity. `PySR` was used in various recent publications addressing symbolic regression in scientific applications [e.g., 101, 177, 256].

TiSR *Thermodynamics-informed symbolic regression* (`TiSR`) is a GP-based framework written in Julia, whose structure is inspired by `PySR` [57]. Its applicability is not limited to thermodynamics, but any kind of problems from the science and engineering domain. It enables multi-objective optimization of different pre-defined or user-specified objectives and is based on the NSGA-II algorithm. The Levenberg-Marquardt algorithm with a lasso regularization method is implemented for constant fitting. `TiSR` allows for fast algorithmic prototyping through simple code structures, while including all state-of-the-art components of a GP-based SR framework. To accommodate problem-specific constraints, further customization options such as weights on data samples to express measurement uncertainties or partially fixed functional structures are provided. The algorithm returns an set of Pareto-optimal solutions in terms of user-defined objective functions.

Further Software Implementations and Code In addition to the previous ones, code for some methods to improve GP as introduced in Sec. 3.2.4 is publicly available. These include, for example, GP-GOMEA [280], the GPOL library [14] that implements GSGP [275], ITEA [93] as well as FEAT [153].

Discussion on Frameworks Overall, the number of available software implementations has grown rapidly, and a single best state-of-the-art framework could not be identified. To compare the performance of different SR frameworks, researchers can submit their

implementation to competitions such as the one held at the Genetic and Evolutionary Computation Conference [91, 152]. For the matter of this thesis, the results on a real-world track with unknown ground truth equations were of interest. Since the datasets had not been known to the competitors in advance, there was no possibility to include domain knowledge in the algorithm to guide the search more efficiently towards desirable and interpretable models. Moreover, the selection of a final model from the Pareto front (for those frameworks that return a Pareto front) was not conducted by a domain expert, but rather by an empirically designed scoring function. Thus, the results of the competition provided limited insights into the framework performance for cases where domain knowledge would have been available and included in the algorithm. In addition to a good performance, the algorithms proposed in this thesis require a framework with extensive customization capabilities to incorporate domain knowledge. Due to this and the rapidly evolving field of GP frameworks, the algorithms proposed in this thesis were implemented in different libraries, namely `DEAP` [90], `PySR` [57] and `TiSR` [170].

3.3 Domain Knowledge in Genetic Programming for Symbolic Regression

As pointed out in Sec. 3.1, physics-informed ML is an emerging area aiming to integrate domain knowledge into the ML process, aligning the resulting models with established physical principles. SR methods are ideal to approach engineering and science problems, as they generate human-readable expressions that inherently provide insights into the model itself. GP is considered the most widely studied method for SR with high customization capabilities, making it a promising candidate for integrating domain knowledge in the context of science and engineering.

For the broader area of ML, there have been a few publications on ontologies that were used to classify different categories of domain knowledge for machine learning observed in the literature [126, 131]. To the best of the author's knowledge, there exists no systematic literature review specifically focused on the integration of domain knowledge in GP. To date, the search for "physics-informed genetic programming" returned only one very few publications mentioning it in their title or abstract [54, 94]. However, there are, in fact, numerous publications that applied GP to both synthetic and real-world datasets and integrated domain knowledge, sometimes without explicitly stating it. The goal of this section is to give an overview of publications in the GP area that integrated domain knowledge in their algorithms and to classify them.

Domain Knowledge Classification

To this end, we adhered to the ontology proposed for physics-informed ML by Karniadakis et al. [126], which aligned well with the patterns observed during the literature review for GP. Rather than attempting to categorize types of knowledge, which is a challenging endeavor even within the realm of philosophical sciences, they concentrated on identifying the types of bias incorporated into the ML process that had been derived from domain knowledge. Three bias types have been identified, namely observational, inductive, and learning bias:

Observational bias is introduced through the data on which an ML model is trained [126]. Examples include the addition of significant data points to guide the algorithm to a desired functional behavior, as well as other methods for data augmentation and feature pre-computation.

Inductive bias refers to prior assumptions that can be incorporated by particular interventions to an ML model [126]. In the case of SR, this includes all measures that restrict the search space, such as the definition of a grammar or a (partially) pre-defined functional form.

Learning bias involves all measures that influence the training process of an ML model to explicitly favor convergence towards solutions that adhere to a desired behavior [126]. Examples include case-specific loss functions, addressing

soft constraints through multi-objective optimization and human interventions during training.

Next to these three bias types, a way of exploiting domain knowledge inherent to multi-objective GP is the selection of an appropriate equation from the Pareto front, typically trading off between accuracy and complexity. This step is crucial for practical applications, where usually only one final equation is required. This *selection bias* is closely related to the field of decision-making, a vast area of research that falls beyond the scope of this thesis.

In the following, we aim to give an overview of GP approaches that have previously employed domain knowledge to introduce bias into their algorithms. Tab. 3.2 summarizes these publications, most of which involve some application from the science or engineering domain. The column "Algorithm" refers to the name of the proposed algorithm in the respective reference, if applicable, or describes the algorithm in own words otherwise. The column "Bias" summarizes how the bias was integrated into the algorithm, which is then classified in the column "Bias Type". Finally, the column "Application/Dataset" specifies the problem on which the proposed algorithms are tested. It becomes apparent that numerous works have proposed ways to integrate domain knowledge into their GP algorithms. The specific application areas are widely spread, though several times the Feynman equations [272] were used as a physics-related benchmark dataset recognized by the research community. A few methods for integrating domain knowledge in GP stand out, as higher attention was given to them in the literature. These will be discussed in the following.

3.3.1 Shape-Constrained Symbolic Regression

Why Shape Constraints?

Various studies considered how prior knowledge in the form of shape constraints can be integrated into the GP algorithm. Shape constraints are useful when datasets cannot cover all important system properties, for example when they contain only few samples, noisy samples, or observation gaps regarding one or multiple variables. When there is domain knowledge available about the expected functional behavior, this can improve the extrapolation capabilities. Potential shape constraints include the monotonicity of a function with respect to a variable, symmetry, convexity, the range of the output values, as well as the slope of a function. These constraints can be addressed using optimistic or pessimistic approximation methods. Optimistic approaches cannot guarantee that a constraint is satisfied, while pessimistic approaches might mark models as infeasible that actually satisfy the constraint.

Feasibility-Based Shape-Constrained GP

Kronberger et al. [144] integrated monotonicity constraints in GP by using *interval arithmetic* (IA) to detect constraint violations and mark individuals as feasible or infeasible. IA was implemented in two solvers: first, an extension of the tree-based canonical GP algorithm that assigns a high penalty to the fitness of infeasible individuals; and second, the ITEA [93] algorithm with separate feasible and infeasible populations. Both algorithms developed models which adhered to the constraints, but demonstrated lower accuracy on the training and test sets compared to other regression algorithms. The authors assumed the cause in the overestimation of model bounds through IA, which might have rejected feasible solutions or have led to premature convergence.

Multi-Objective Shape-Constrained GP

Another approach more frequently studied in the literature is multi-objective shape-constrained GP, where each constraint is addressed as a separate objective, or all constraints are combined into one objective with the constraint violation to be minimized. Bladek and Krawiec [24] integrated shape constraints through counterexample-driven GP, which is an optimistic approach employing a *satisfiability solver modulo theories* (SMT) solver to detect constraint violations and extend the training data with counter examples. The error on the counter-examples set was optimized as a separate objective. Haider et al. computed constraint violations with IA and minimized the violations with NSGA-II [104], *multi-objective evolutionary algorithms* (MOEA)/D [104] and NSGA-III [107]. In [106], Haider et al. compared an optimistic approach, which

Ref.	Algorithm	Bias	Bias Type	Application/Dataset
[12]	Hybrid Symbolic Regression	Fix top structure of the model tree	Inductive	Metal Bending Process
		Precomputed, dimensionless features	Observational	
[24]	Shape contraints using an optimistic approach	Counter-example driven SR with SAT solver	Learning	Gravity, Resistance2, Resistance3
		Counter-example driven SR with computing constraint violations on a constraint dataset and minimizing constraint violations with NSGA-II	Learning	
[26]	Implicit Symbolic Regression	Problem-dependent fitness measure including time derivatives	Learning	Plasticity models
[53]	Dimensionally aware GP	Dimensional consistency is enforced using grammars	Inductive	Higgs dataset, DVCS dataset, $\tau\mu 3$ - dataset
	Transition Matrix	Expert-defined transition matrix with probabilities to choose the next operator given the current operator, also forbids certain combinations	Inductive	
[57]	Dimensionally aware GP including constants with unknown units	High penalty value for individuals non-compliant with units	Learning	No appplication given
[60]	GNN as inductive bias for GP	GNNs with internally separable structure to approximate interactions between entities	Inductive	Newtonian Dynamics, Hamiltonian Dynamics, Cosmology
[73]	Dimensionally aware GP	Constrained Initialization Procedure and Local Dimensionally aware Search	Inductive	Feynman
[94]	Physics-informed genetic programming-based symbolic regression (P-GPSR)	Integration of requirements towards a symbolic model, which were derived from a thermodynamic-based analysis, through objective functions	Learning	Gurson yield function from thermodynamics
[104]	Multi-objective shape constraints	Multi-objective approach with NSGA-II and MOEA/D as algorithms and Interval Arithmetic to compute constraint violation	Learning	Feynman
[105]	Presentation and application of their previously introduced shape-constraint algorithms in [104, 107, 106] to real-world datasets			Twin-Screw Extruder Modeling, Data Validation for Industrial Friction Performance Measurements, Magnetization Curves
[106]	Shape constraints with pessimistic approach	Interval Arithmetic to evaluate shape constraints	Learning	Feynman
	Shape constraints with optimistic approach	Domain-specific inclusion of significant data points	Observational	
[107]	Shape constraints with NSGA-III	Multi-objective approach (NSGA-II and NSGA-III) to minimize constraint violations which are detected using Interval Arithmetic	Learning	Feynman
[109]	Replacement Mutation and Adaptive Replacement Mutation	A-priori knowledge about subprograms: Knowledge archive of subprograms which is used during mutation	Learning	Program Synthesis Benchmark Suite (PSB1) and Composite Problems constructed from PSB1
[111]	Grammatical Evolution for Program Synthesis with varying grammars	Automatic mapping between problem description and built-in functions and increasing the likelihood of these functions	Inductive	Program Synthesis Benchmark Suite
		Problem-dependent fitness functions that measure how many of expected program elements are included	Learning	
		Extracting significant constants, operators and not-to-use elements by human from problem description and updating grammar	Inductive	
[127]	Linear Scaling	Linear Scaling to scale the output of a program to the target output, if shape is well, but scale is wrong	Inductive	Collection of SR problems from other papers that present improvement methods for GP
[128]	Dimensionally-aware GP computes the number of unit violations in an individual	GP with brood selection, only the best individual w.r.t. the cheaper dimensional analysis survives	Learning	Bernoulli equation for energy conservation
		Multi-objective GP	Learning	
		Multi-objective GP with brood selection	Learning	
[130]	Scientist in the loop approach providing different integration points where the human can insert problem knowledge	Division into distinct feature groups using user input and AutoML	Observational	5 Short equations with different characteristics (large number of features, complex function shape, unknown ground truth, noise)
		Loss function selection	Learning	

Ref.	Algorithm	Bias	Bias Type	Application/Dataset
		Allocation of feature weights (to single features and not to data points)	Observational	
		Steering of sampling strategy for new solutions through directing it towards using specific features more often or exploring equations of a specific complexity	Learning	
		Equation structure constraint	Inductive	
[142]	Knowledge-guided Genetic Improvement by combining Tree-based GP and Grammar-guided GP	Operator graph contains concepts in a language and knowledge associated with it	Inductive	No experiments, only proposed this approach
[144]	Single-objective Interval Arithmetic for SC	GP with Interval-arithmetic to analyze SC, high penalty for constraint violations	Learning	Feynman
	Feasible-Infeasible Two Population with Interaction-Transformation	Feasible population minimizes error function, infeasible population minimized constraint violation	Learning	
[146]	Inclusion of categorical features	Grouping of material properties into factor variables	Observational	Friction systems with varying materials
[148]	Shape constraints for symmetry w.r.t. arguments or w.r.t. domain, output range, function monotonicity, function slope using an optimistic approach	Counter-example driven SR with computing constraint violations on a constraint dataset and minimizing constraint violations with NSGA-II	Learning	Turtlebot, Drone, Magnetic Manipulator
[149]	Shape constraints for symmetry w.r.t. arguments or w.r.t. domain, output range, function monotonicity, function slope using an optimistic approach	Counter-example driven SR with computing constraint violations on a constraint dataset and minimizing constraint violations with NSGA-II	Learning	Resistance2, Magnetic Manipulator, Pressure
[162]	GNN as inductive bias for GP	Relative Mean Weighted Error as problem-dependent loss function	Learning	Orbital Mechanics
		Transformation to spherical coordinates, data augmentation through rotation	Observational	
		GNNs with internally separable structure to approximate interactions between entities	Inductive	
[163]	Dimensionally Aware Multi-Objective GP (DAMOGP)	Multi-objective GP, dimensional analysis uses prime numbers to express a unit, unit-adjusted evolutionary operators	Learning	Real-world crowd datasets
[174]	MOJITO to include hierarchical domain-specific building blocks	Set of expert-specified, trusted, hierarchically organized analog building blocks, organized as a parameterized context-free grammar	Inductive	Analog circuit topology synthesis
[176]	Constrained Dimensionally-aware GP (C-DAGP)	Computation of a dimension gap, which is weighted with an adaptive penalty coefficient and added to the error measure	Learning	Dynamic Job Shop Scheduling Task
		Culling with crossover size of two and selection of the offspring with lowest dimension gap	Learning	
[196]	Physics-regularized fitness function for analytical solutions to differential equations	Fitness function is augmented with a measure of how well an equation satisfies the prescribed differential equations	Learning	Euler-Bernoulli Differential Equation, Poisson's Equation
[209]	SC and Extended Constraints (EC)	Define soft constraints computed on conversions of an expression using additional features, such as f(x)/h. Violation added to error objective	Learning	Magnetization Curves
[211]	GNN as inductive bias for micro-macro evolution GP	GNNs with internally separable structure to approximate interactions between entities	Inductive	Collective behavior: Hexagonal Shape Formation, Square Shape Formation, Coordinated Motion
[218]	Dimensionalization through formal grammars	Parts of the function are fixed in the grammar definition	Inductive	Force-time relation of mechanical indentation tests
		Dimensional constraints implemented in the grammar	Inductive	
[219]	Dimensionalization through formal grammars	CFG-compliant initialization procedure to overcome diversification and bloat issues in initialization using CFG	Inductive	Kelvin-Voigt, four-element model

Ref.	Algorithm	Bias	Bias Type	Application/Dataset
[234]	Hybrid Symbolic Regression	User-defined special operators: spatial differential operators	Inductive	Ocean subgrid parametrization for eddy and jet configurations
		Human in the loop to remove/add/edit equations	Learning	
[236]	Multiview Symbolic Regression	Fitting different coefficient values within structurally equal expressions to multiple datasets from the same problem family, but with different experiment set-ups	Learning	Chemistry, Finance, Astrophysics
[247]	Seeding methods to incorporate expert knowledge of partial solutions	Approximate Equation Seed inserts exact solution, with one subtree being replaced by a constant	Inductive	100 randomly generated target equations of varying complexities
		Shuffled Equation Seeding uses approximate equation and exchanges two random subtrees	Inductive	
		Building Block Mutation where each subtree of the approximate equation is a building block	Learning	
[249]	EvoStencils	Definition of a problem-specific grammar with problem-specific functions	Inductive	Construction of efficient multigrid solver
[250]	Dimensionalization through formal grammars	Dimensional constraints implemented in the grammar	Inductive	Materials behavioral law
	Examples for domain knowledge on different levels	Problem-dependent representation based on Voronoi diagrams	Inductive	Seismic Underground Prospection
		Problem-dependent geometrical crossover operator for Voronoi representation of a solution	Learning	
		Problem-dependent Fitness function that uses a proxy measure	Learning	
[252]	Joint hand-designed and GP-generated equation	Combination of Phenomenological model (fixed equation part) and an empirical correction term found through GP	Inductive	Resistance Spot Welding
[278]	Comparison of influence of different domain knowledge insertions	Pre-computed features	Observational	Five different datasets describing plastic deformation of copper
		User-defined weighting of important samples in the dataset	Observational	
		Insertion of new data points into gaps	Observational	
		Forcing candidate solutions to be above zero by wrapping it into an exponential function	Inductive	
[291]	Biasing GP through CFG	Context-free grammar to limit the search space to desired solutions	Inductive	
[299]	Dimensionally aware GP	Multi-objective GP to minimize the number of unit violations next to an error objective	Learning	Three benchmark equations from the Stokes flow past a single spherical particle with and without pre-computed features
	Assessment of feature pre-computation derived by domain knowledge	Apply trigonometric functions to angle features, and square, cube and unary division to all other features	Observational	

Table 3.2: An overview and classification of the different types of domain bias in GP algorithms found in the literature.

added domain-specific significant data points to the dataset, with a pessimistic approach using IA to estimate the bounds of a model and its partial derivatives. While the optimistic approach indicated better extrapolation capabilities, the pessimistic performed better on noisy datasets. Kubalik et al. employed the counter-example driven constraint computation and optimized it with NSGA-II and multiple regression GP [149]. They extended this approach with a feature mixing technique in another publication [148].

Shape-constrained GP algorithms found applications in various areas, as further studies by Winkler et al. [105] and Piringer et al. [209] demonstrated. Multi-objective approaches generally allow for the development of infeasible models, and still yield partially feasible models when some constraints are extremely challenging to learn. This makes them less restrictive compared to methods that enforce the constraints. However, further research on how the inclusion of constraints in GP affects the search space and population diversity would be interesting [144]. Methods to include shape constraints in ML models were also studied outside the SR realm, for example in [103, 286].

3.3.2 Unit-Aware Genetic Programming

The following subsection is largely based on the author's publication [224].

Why Dimensional Analysis?

Another area of significant research is the development of symbolic models that align with physical units. An established approach in engineering sciences is the Buckingham Π theorem [33], which transforms variables and constants into dimensionless quantities through multiplicative operations. While making variables dimensionless preserves physical consistency, it also eliminates the associated units. As studied in more detail by Tenachi et al. [267], unit information can guide the SR algorithm in finding an appropriate functional form. Thus, removing unit information could also cause the loss of valuable constraints on possible functional forms.

In addition, domain experts in science and engineering frequently require solutions to adhere to physical laws and avoid unit-violating operations, such as the addition of a quantity in meters and a quantity in seconds. The acceptance of results of SR algorithms can suffer, when solutions do not stick to expected physical laws, with unit-conformal operations being one of the critical requirements. The literature on unit-aware SR uses the terms "unit" and "dimension" inconsistently. In the following, these terms will be used interchangeably. It should be noted that "dimension" only refers to the dimensionality of the data when explicitly mentioned, such as in phrases like "high-dimensional" or "six-dimensional".

Early Research

The consideration of physical units in the search for symbolic models was studied early in the GP area. Keijzer and Babovic suggested different methods to handle unit violations in GP [128]. The dimensions of a feature were expressed as a vector of real-valued exponents corresponding to the units of length, time and mass. A velocity feature in meters per second was thus expressed as $[1, -1, 0]$. A goodness of dimension criterion computed the distance from the desired dimensions by summing over the number of required transformations within the expression to achieve the desired dimension. They furthermore introduced a repair function that corrected unit violations by simple multiplication with a constant. This constant had a value of 1.0 and transformed the units of the variables into units valid for the specific operation. Their experiments aimed at recovering the Bernoulli equation for energy conservation. The results suggested that a multi-objective approach minimizing the goodness of dimension criterion yielded the best results. Furthermore, unit information gained importance as the noise level of the data increases. Keijzer and Babovic [128] also emphasized the importance of constants within the dimensional analysis. The algorithms with dimensional analysis only found the ground truth solutions regularly when the used constants along with their units were given as input features. However, it needs to be noted that their algorithm in 1999 did not include parameter fitting, such as current state-of-the-art algorithms. Instead, new randomly generated constants were assigned a dimensionless unit vector of $[0, 0, 0]$, which may have prohibited the algorithm from finding unit-conformal equations with suitable parameters. Overall, this paper laid the foundations for future research in this area.

Unit-Aware GP Without Unknown Constants

Some methods from the literature enforce dimensional consistency by defining a building grammar, for example for unit-aware feature construction in experimental physics [53] or predicting material's response to a mechanical test [218, 219, 250]. Similarly, strongly-typed GP ensures dimensional consistency by defining units instead of types, such as studied in [74]. Compared to other constraint handling methods presented in the next paragraph, grammar-based and strongly typed GP require a higher implementation effort. Ideally, this inductive bias restricts the search space in a way that feasible solutions can be explored more efficiently, which in turn leads to faster algorithm convergence. However, as Mei et al. [176] pointed out, the resulting search space can constitute multiple disconnected feasible regions, which can be challenging to reach. This may result in convergence towards local optima.

Other approaches allowed the algorithm to evolve equations which contain unit violations, and put a learning bias in place to reduce them. Overall, three predominant ways for unit-related constraint handling in GP can be identified in the literature. First, a multi-objective variant that minimizes the number of unit violations as an additional objective [163, 299]. Li et al. [163] encoded each of the seven SI unit as a different prime number, on which dimensional analysis propagated through the operations of the tree. Repeated mutation of unit-violating subtrees led to overall fewer infeasible solutions. Those individuals that nevertheless produced a dimension error were assigned a dimensional inconsistency measure, which was optimized in a multi-objective way. Zille et al. [299] encoded units as a vector of exponents and computed the dimension error as the sum of all unit-violating operations within an individual. The Manhattan distance between the output and target unit vector was furthermore added to the dimension error.

Second, a repair mechanism that manipulates the operations to match the input and target units, for example, by multiplication with a constant to balance units [128].

Third, the addition of a penalty term to the primary error objective to account for unit violations. The most drastic case studied in the literature was the "death penalty", which assigned a large penalty value to ensure that an individual did not survive to the next generation [17, 176]. The brood selection strategy of [128] shared similarities with the death penalty approach: Multiple offspring were generated from one individual, and the one with the smallest unit violation was added to the population. This strategy was already applied in the reproduction stage and not in the subsequent survival selection stage of an algorithm.

Unit-Aware GP with Unknown Constants

The development of new empirical equations from real-world datasets creates additional difficulty due to newly generated constants: the amount, value, and position within the equation as well as the units of the constants are not known beforehand. Every constant can take on arbitrary units, which makes previous dimensional analysis in unit-aware GP approaches impossible to use. However, both, unknown constants and symbolic models that adhere to physical unit constraints, are an important requirement of domain experts from various scientific fields. To overcome this issue, the `PySR` backend `SymbolicRegression.jl` recently released a functionality to consider unknown constants in the dimensional analysis [57]. An equation is evaluated, and the units are propagated through the equation accordingly. Constants act as so-called "wildcards" and can take on arbitrary units. In case of unit violations, a penalty value is employed. This penalty does not account for the number of unit violations, i.e., solutions with few violations are treated equally to solutions with many violations. A weighted sum of the primary objective and the penalty term is built to penalize unit violating individuals. Building a weighted sum of these two measures is also challenging, as the scales are different, and an additional weight parameter needs to be specified. Depending on the penalty value, which is large in the standard settings of `PySR`, this can have the effect of a death penalty.

However, combining parameter estimation with the death penalty for constraint handling has the negative effect that a solution, that does not survive to the next generation because of the death penalty, still uses computational resources for the expensive parameter estimation. This is, to the best of the author's knowledge, the only approach to conducting dimensional analysis on equations with unknown constants in the GP area. Using the unit propagation scheme as proposed in `SymbolicRegression.jl`, it can be interesting to assess the performance of other constraint handling methods.

3.3.3 Further Approaches to Integrate Domain Knowledge

Assumptions About Function Structure

Another way of inducing domain knowledge into the GP algorithm is to make assumptions about the expected functional structure and enforce it in the ex-

pression. Asadzadeh et al. [12] fixed the top structure of the GP tree so that evolution takes place on the lower levels. Linear scaling as presented in [127] also posed a strong inductive bias of constant placement on the functional form, with certain drawbacks as earlier discussed in Sec. 3.2.2. Schwab et al. [252] based their final expression on a phenomenological model from the literature and developed only the correction term with GP. Versino et al. [278] wrapped the develop expression into an exponential function to guarantee positive output values. Seeding the population with the expected functional form or predefined subtrees is a less restrictive way to provide an initial guess about the expression, as studied in [247]. This gave the algorithm the chance to converge to a better model without any of the defined subtrees, in case such a model exists.

GNN as Inductive Bias

A very specific functional form applicable to various problems is enforced when GNNs as inductive bias are used. Cranmer et al. [60] proposed this method for high-dimensional physical systems with interacting entities, as previously discussed in Sec. 3.2.5. The underlying functional form was restricted to the summation of pairwise interactions, leading to a nested overall function $y = g(f(x))$. This method was applied to Hamiltonian and Newtonian dynamics with known ground truth, as well as real-world datasets from cosmology. A similar approach was applied to model collective behavior in swarms in [211], where the interactions between swarm members were first approximated by a GNN. Lemos et al. [162] used this method to rediscover the orbital mechanics of the solar system. They included further domain-specific learning bias, such as the relative mean weighted error as a loss function, to account for large dynamic ranges in the dataset. Observational bias was integrated by transforming the coordinate system to spherical coordinates centered around the sun, and augmenting the existing data by rotation of the planet arrangements around the center.

Pre-Computed Features and Problem-Specific Functions

A way to encourage the algorithm to use expected combinations of operations and features is to add pre-computed features to the terminal set, as well as defining problem-specific functions. The exact operations usually depend on the specific problem and thus vary across the publications that employ it. Schmitt et al. [249] defined a problem-specific set of functions and the respective grammar to evolve efficient geometric multi-grid solvers. The approach generated models that were competitive or more efficient than previous methods from the literature. Zille et al. [299] added precomputed features to the terminal set to reduce the overall number of operations required in the final expression predicting the Stokes flow around a spherical particle. Trigonometric functions were applied to features that represent angles, and square root, square and cube operations were applied to all other features. A similar approach was used in [278] to model the plastic deformation of copper. Dimensionless features were pre-computed and added to the terminal set in [12] to describe a metal bending process. He et al. [109] proposed a replacement mutation operator and provided a knowledge archive of subprograms which were inserted into the GP trees during mutation. In [111], an additional objective was employed to maximize the frequency of expected problem-specific subprograms within an expression. A set of expert-specified, trusted, hierarchically organized building blocks were proposed in [174], which were implemented as a parameterized context-free grammar. The method was used to evolve analog circuit topologies. Ross et al. [234] implemented a special spatial differential operator which was required to predict eddy and jet configurations in oceans.

Human Interventions

Engaging the domain expert during the expression search and offering human interventions allows for a more flexible guidance of the search process and integration of domain preferences. In [130], the applicability of existing SR methods for physical systems was discussed. The `SciMED` framework was proposed, a scientist-in-the-loop approach, to include prior knowledge in the search for useful equations. Human input was required to divide the features into distinct groups and remove redundant features, to select an appropriate loss function, and to assign higher weights to features that are expected to play an impor-

tant role in the final equation. Moreover, the domain expert could steer the sampling strategy for new solutions during evolution and direct the algorithm towards exploiting models of a specific complexity. Their method outperformed AI Feynman as well as GP-GOMEA [280] in some cases. In [234], the domain expert could remove existing or add new equations at specific points during evolution, and was also allowed to edit existing equations. In this way, expert preferences were not only conveyed before and after algorithm execution, but also during the search itself.

Other Constraints and Guidance Measures

Generally, GP-based SR approaches provide the opportunity to include prior knowledge on various levels. An attenuated variant of a more restrictive grammar is to define building constraints. Popular frameworks such as `PySR` allow for the definition of building rules, for example defining a maximum length of the input argument to a function, or for preventing the nesting of operators [57]. However, these constraints are mainly applied during the initial generation of solutions and may be violated through mutation and crossover. Cherrier et al. [53] presented an expert-defined transition matrix with probabilities to choose the next operator given the current operator. This matrix also prevented certain combinations of operators.

Multiple measures to evolve problem-specific equations adhering to the expectations of domain experts were presented in [250], with the goal to model seismic underground prospection. In addition to enforcing dimensional constraints through formal grammars, the algorithm also included a problem-dependent representation based on Voronoi cells, respective crossover operators, and fitness functions.

Versino et al. [278] also studied several ways to systematically integrate domain knowledge about material properties and functional forms into the GP algorithm. Observational bias was introduced by the pre-computation of features, weighting the influence of important data samples during evaluation, as well as adding artificial data points to the dataset, which the domain expert was confident represent the expected functional form. To guarantee output values above zero, they imposed an inductive bias on the model by wrapping the expression into an exponential function. Furthermore, they provided approximate equations developed by domain experts to seed the initial population of models. They concluded that the measures to integrate domain knowledge helped the algorithm to identify equations that were in some cases on par with other models in the literature. They furthermore stressed the limits of GP methods without integrated domain knowledge, such as highly unpredictable model behavior deviating from physical principles, overfitting to training data, and reproducibility problems due to the stochastic nature of GP.

3.4 Summary of this Chapter

In this chapter, the related works in terms of PIML and SR, with a focus on recent developments in the GP realm, have been presented and discussed. Physics-informed neural networks and symbolic regression methods outside the EC area have been introduced in Sec. 3.1, which are often tailored to model specific physical systems.

Several aspects relevant for the development of meaningful symbolic models for science and engineering applications employing GP have been covered in Sec. 3.2. These are, generalization, constant fitting, distributed algorithms, high-dimensional GP, software implementation, as well as other algorithm components to improve performance. For each aspect, recent improvements that render GP a suitable approach for science and engineering problems have been highlighted, as well as limitations and gaps in the literature.

The integration of domain knowledge plays a crucial role in the development of useful models for science and engineering problems. In Sec. 3.3, an overview and

classification of methods in the literature to bias GP algorithms through domain knowledge have been presented. These algorithms are used in a wide range of applications within the science and engineering domain. Shape-constrained GP and unit-aware GP have been identified as notable trends in the literature, with various publications and different approaches proposed.

Overall, despite the advances improving GP for science and engineering problems, several limitations have been identified, which build the motivation for this thesis. Some of these limitations will be addressed in the remainder of this thesis:

- Island models are a method to improve the model accuracy and success rate of GP algorithms, which are both important requirements for domain experts. In recent literature, mainly single-objective regression problems have been studied for GP [198, 199]. However, further research is required to assess the application of island model GP to multi-objective problems and its effect on the success rate of the algorithm.

- Graph neural networks are a well-motivated inductive bias to tackle high-dimensional SR problems that can be modeled as interacting entities and make them tractable for GP. Studies mainly demonstrate how this method is capable of recovering existing relations [60, 162, 211], and has only been used in a few publications to identify models for problems with unknown ground truth and compared to handcrafted equations from earlier research [e.g., 60, 279]. Assessing its considerable potential for problems from the science and engineering domain to develop new symbolic models, as well as enhancing the existing approach with other types of domain knowledge, deserves further attention.

- Enforcing unit constraints and dimensional analysis in GP is a crucial requirement for domain experts, particularly when developing new data-driven models. While existing research [163, 176, 272] typically assumes that constants and their units are known beforehand, this assumption does not hold true in cases where new constants with unknown units are learned. Further investigation is needed to address how GP can be adapted to enforce unit consistency even when these unknown constants are introduced.

- Scalable problems are essential for testing the robustness and adaptability of symbolic regression methods across varying levels of complexity. The related studies presented in Sec. 3.3 apply their methods to various problems. However, systematic and scalable methods for applying domain knowledge to complex, real-world problems have yet to be fully explored. The application of algorithms to problems that are inherently scalable and varying in complexity is required, to evaluate how different methods of incorporating domain knowledge can be applied across problems of varying difficulty and disciplines.

Part II

Problem Definition and Initial Approaches

4 Inverse Kinematics of Robotic Manipulators

Introduction

This chapter introduces a real-world problem from the field of robotics to serve as a benchmark for evaluating existing methods in GP when applied to complex, domain-specific problems with unknown ground truth. By demonstrating the application of state-of-the-art techniques before introducing the advancements presented in this thesis, this chapter provides a comparative baseline for the contributions discussed later in this thesis.

This chapter is largely based on the author's publications [221, 226]. The author thanks Dr.-Ing. Christoph Steup and Pravin Pandey for providing datasets, domain knowledge and expert insights about robotic manipulators and inverse kinematics, and for critically reviewing draft versions of this chapter.

4.1 Inverse Kinematics Problem

Robotic Manipulators

Robotic manipulators are at the center of process automation. Most applications today, like automatic welding or pick-and-place tasks, require these manipulators to operate with high flexibility in movement [289]. At the same time, in special use cases with many manipulators in a small arena, like in swarm robotics, we need them to operate in very confined spaces. As a result, a plethora of robotic manipulators has been constructed by many companies, which do not follow the standard six *degrees of freedom* (DOF) configurations. One example is the 5 DOF *KUKA youBot* manipulator that serves as a use case for this chapter. By having one joint less than a standard six DOF manipulator [150], this robot takes up less space and is still quite flexible. While these robots fulfil the flexibility and compactness criteria, atypical joint configurations complicate the kinematic analysis because the standard methods cannot be applied anymore. Consequently, understanding the kinematics, i.e., the relation between the DOF and the resulting movement of the rigid body of an arbitrary robotic manipulator, is crucial [226].

Manipulator Structures and Joint Types

The general structure of a robotic manipulator, similar to the one depicted in Fig. 4.1, is an open kinematic chain. It consists of multiple, typically rotational joints, which are connected by links. Non-rotational joints, such as prismatic joints, are left out of consideration in this thesis. The joint rotation parameters $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_\omega]$ are the variables of interest, with the number of joints, or DOF, denoted by ω. In general, a robotic manipulator operates in a six-dimensional environment $\mathbf{X} = [\mathbf{p}, \mathbf{o}]^T = [x, y, z, \Phi, \Theta, \Psi]^T$. In this representation of the pose $\mathbf{X}$, the vector $\mathbf{p} = [x, y, z]^T$ corresponds to the position of the end-effector in Cartesian space, and $\mathbf{o} = [\Phi, \Theta, \Psi]^T$ represents the end-effector orientation

Figure 4.1: KUKA youBot as available in the swarmLab for robotics at Otto-von-Guericke University, Magdeburg.

using three Euler angles [121].

Forward Kinematics

The goal of the *forward kinematics* (FK), denoted by g, is to determine the pose of the end-effector given the robot joint angles, i.e., $\mathbf{X} = g(\boldsymbol{\theta})$. The calculation is fairly straightforward since there is exactly one pose for each combination of joint values, which may be feasible or not. In other words, there is a possibility that the FK results in self-collisions of the manipulator or collisions with the ground. A common standard to describe the kinematic model of a robot is the *Denavit-Hartenberg* (D-H) convention, which requires only the four following parameters per joint [121]:

1. Link length a_i: The distance between two consecutive joint axes along the shortest distance between the two axes. It represents the length of the link along the x-axis of the current joint frame.

2. Link twist α_i: The angle between the z-axes of two consecutive joint frames, measured around the x-axis. It defines the orientation of the two links relative to each other.

3. The joint distance or link offset d_i: The distance along the z-axis between two consecutive joint axes. It represents the displacement along the z-axis of the previous frame to the common normal.

4. The joint angle θ_i: This is the angle around the z-axis between the previous link and the current link, which represents the rotation required to align the x-axis of the previous joint with the x-axis of the current joint.

For any given rotational link, three of these quantities are constant, while only θ_i is variable and defines the actual movement [121].

Inverse Kinematics

The *inverse kinematics* (IK) problem aims to find the robot angles required to achieve a specified target pose, i.e., $\boldsymbol{\theta} = g^{-1}(\mathbf{X})$. In contrast to the FK, there are multiple solutions to the *inverse kinematics* (IK) problem. This makes the problem considerably more complex than the forward kinematics. In general, finding a closed form solution is of great interest, i.e., finding an explicit relationship between a joint variable θ_i and the elements of the six-dimensional pose vector $\mathbf{X}$. Since multiple valid joint configurations can be found for a given pose, closed-form solutions allow for real-time movements and can furthermore provide decision rules to prefer a particular configuration over another [121]. Moreover, an explicit relationship enables the mathematical analysis of safety-relevant real-life scenarios. Various closed-form solution approaches have emerged for different applications within the last decades [9]. Classical analytical approaches provide IK solutions in real-time. However, they are often not applicable to robots with non-standard axis configurations, since such robots do not always have unique solutions to the IK problem. Modern computational approaches like ANNs can overcome this issue, but lack transparency and explainability. Balancing these conflicting requirements, finding an accu-

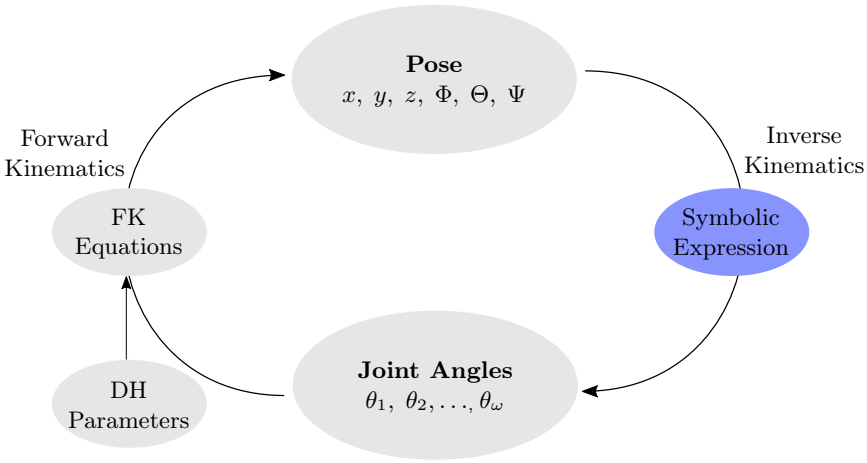


Figure 4.2: Illustration of the inverse kinematic problem and the role of the GP algorithm within the problem setup [221].

rate, real-time capable and explainable IK model for a non-standard configured robot is a complex task [226].

Chapter Goals

In this chapter, we aim to develop symbolic models for the inverse kinematics of a non-standard robotic manipulator. We used GP to evolve human-readable equations that could be executed in real-time, while providing explainability and allowing the adaptation to non-standard robotic configurations. To achieve this, we incorporated GP alongside the FK equations in the evaluation function, as illustrated in Fig. 4.2.

Chapter Overview

In the following, an overview of the related work in the literature, with a focus on data-driven methods employing GP and ANNs, will be provided. We then introduce the novel *cooperative coevolutionary genetic programming for inverse kinematics* (IK-CCGP) approach, which tackles the need of an IK model for multiple outputs, i.e., one value per joint of the kinematic chain. Domain knowledge is integrated at various levels across the entire pipeline of the algorithm. The proposed GP algorithm was examined on a *KUKA youBot* with 5 DOF, which is available and frequently used for various tasks in the robotics laboratory *swarmLab* at Otto-von-Guericke University, Magdeburg. We performed several experiments on various settings of the problem, and compared the results with the reported results of ANN-based approaches in [5, 262]. Our experiments suggest that this approach achieves competitive results in certain areas of the workspace. However, ANN-based approaches consistently provide more accurate solutions with smaller failure rates. Although the evolved models allow for interpretation, several drawbacks of the proposed method could be observed, which motivated further contributions in this thesis.

4.2 Related Research

Genetic Programming Approaches

Approaching the IK problem with GP is mainly driven by two early publications by Chapelle et al. [39, 40]. In both publications, the problem was modeled in a single-objective way with a length-penalizing RMSE of the joint angle as a fitness function. All joint equations were learned sequentially starting from joint one, feeding the result to the next joint and so forth. The proposed setting reached an accuracy of 10^{-2} radians for the first joint θ_1 and 10^{-1} radians for the last joint θ_6. The maximum error on the tested instances was 0.3 radians, which is a deviation too large for the application in real-world scenarios.

In addition to the mentioned works, the principles of evolution were mostly applied to find off-line, i.e., not real-time capable, IK solutions for given robotic manipulators. Parker et al. [204] proposed an EA model that approximates the joint values for a given end-pose. This approach incorporated the FK equations and used the positioning error between the desired and the learned poses as

the objective to be minimized. Kalra et al. [123] used EAs to find multiple solutions for the IK problem.

Another application of GP in the robotic field is the calibration of manipulators. Dolinsky et al. [71] used GP to approximate an error function between the desired and the actual position of the end-effector caused by mechanical inaccuracies of the real-world robot compared to its simulation. They proposed a coevolutionary algorithm, as multiple joint corrections need to be calculated at the same time, which all mutually affect each other.

Artificial Neural Network Approaches

ANNs have been applied to the IK problem for simple manipulators [76] and six DOF standard manipulators [63]. These approaches solely give the target pose as an input to predict the corresponding joint angle values. They mainly focus on tuning the parameters of the model, such as the number of hidden neurons or the learning rate. Almusawi et al. [5] proposed to include the current joint configurations of a six DOF robot in the input parameters, which improved accuracy, especially for continuous trajectories such as drawing a circle with a pen.

A different error measure was suggested by Srisuk et al. [262]: Instead of comparing the learned and desired joint angles of the 4 DOF manipulator, the authors implemented the FK equations to compute the end-effector pose from the learned angles. The error function was thus a distance measure between the learned pose and the desired pose. By incorporating the FK equations, this approach allows learning the underlying kinematic behavior without limiting good results to pre-defined joint angles. This helps to overcome the singularity problem of a robot, where multiple joint configurations lead to the same end pose.

In a comparative study, El-Sherbiny et al. [257] evaluated different modern IK approaches, namely ANNs, adaptive neuro fuzzy inference systems (ANFIS) [67] and genetic algorithms, to solve the IK problem for a non-standard 5 DOF manipulator. The ANN and ANFIS approaches predict unseen data instances with a position MSE of 0.0016m, which can be roughly translated to 4cm mean absolute position error. Wagaa et al. [285] recently compared different ANN architectures to solve the IK problem, namely a feed-forward NN, a convolutional neural network, long–short-term memory, gated recurrent unit, and bidirectional long–short-term memory ANN. The latter outperformed all other methods for six-DOF manipulators. However, all the tested networks had massive sizes with millions of trainable parameters and were trained on a comparatively small dataset size. This combination of settings elevates the risk of overfitting, and the test set contains, in the worst case, only 50 samples, which cannot represent the workspace reasonably well. A comprehensive overview of the plethora of ANN-based IK approaches can be found in [9, 285].

Graph Neural Network Approaches

Recently, the identification of manipulator-like dynamical systems has been approached with *graph neural networks* (GNNs). While this area is still in its early stages, a few remarkable publications on this topic are available. In a 2018 study, Sanchez-Gonzales et al. [243] proposed an object- and relation-centric representation of six complex 3D physical systems to perform prediction, inference, and control tasks. Links are regarded as "bodies" and encoded as nodes of a graph, and joints as edges or connections between bodies. Their GNN-based approach achieved competitive performance in trajectory planning of a robotic arm, but did not solve the problem of accumulated errors over longer trajectories. Furthermore, the approach performed well on systems with shared structural characteristics, such as a swimmer or manipulator, but could not fully unfold its potential on the "cheetah" structure with few opportunities for sharing. Interestingly, Kim et al. [132] proposed an opposite encoding than [243], with joints translating to nodes in the graph and links as edges connecting the joints. They used an encoder-decoder architecture with two embedding spaces for structure and pose. To learn the IK model, the structure embedding and end-effector positions were the input to the pose embedding, from which the joint angles required to reach a certain position were reconstructed.

Furthermore, different message-passing and connectivity schemes within the graph were compared. The proposed model performed similarly to a baseline multi-layer perceptron on the IK task.

4.3 Proposed Methods

In the following, we first present our approach to modeling the IK problem using several objective functions. Afterward, we introduce the proposed universal, problem-independent algorithm IK-CCGP.

4.3.1 Objective Functions

To reach a certain pose in Cartesian space, two different error functions can be utilized during the training of GP, which both lead to the same final pose when learned perfectly: the joint angle error $f_1(\theta)$, and the pose error $\mathbf{f}_1(\mathbf{X})$, where θ represents the angle and $\mathbf{X}$ the pose. In addition, we considered other objective functions such as correlation coefficient and dimension penalty, as first introduced by Zille in [299].

Angle Error vs. Pose Error

Error Objective (f_1): $f_1(\theta)$ describes the difference between the given and the produced joint value in radians, and is to be minimized for each of the joints in the kinematic chain. This objective function is designed to enforce GP to learn the exact angle values given in the training data to reach a certain pose. Hence, a genetic program is to be identified that transforms the input pose into the desired joint angle value by optimizing for $f_1(\theta)$:

$$f_1(\theta, \hat{\theta}) = \sqrt{\frac{1}{\kappa}\sum_{i=1}^{\kappa}(\hat{\theta}_i - \theta_i)^2} \tag{4.1}$$

where κ refers to the number of data points involved. An alternative to $f_1(\theta)$ is $\mathbf{f}_1(\mathbf{X})$. Due to the kinematic setup of a robotic manipulator, multiple joint configurations can lead to the same end pose, i.e., more than one valid solutions exist for the given problem. While the $f_1(\theta)$ error function limits GP to precisely learn the target joint values to reach the desired position, the $\mathbf{f}_1(\mathbf{X})$ error function allows for more flexibility in the learning process. Instead of comparing each learned joint value to its given target value, their resulting pose is computed using the FK equations and compared to the desired pose. We considered the position and orientation of a pose as separate objectives since they also come in different units. Hence, the optimization problem can be formulated using two objectives:

$$\begin{aligned}\mathbf{f}_1(\mathbf{X}, \hat{\mathbf{X}}) &= \left[f_1(\mathbf{p}, \hat{\mathbf{p}}),\ f_1(\mathbf{o}, \hat{\mathbf{o}})\right]^T = \\ &= \left[\sqrt{\frac{1}{\kappa}\sum_{i=1}^{\kappa}||\hat{\mathbf{p}}_i - \mathbf{p}_i||_2^2},\ \sqrt{\frac{1}{\kappa}\sum_{i=1}^{\kappa}||\hat{\mathbf{o}}_i - \mathbf{o}_i||_2^2}\right]^T\end{aligned} \tag{4.2}$$

where $\mathbf{p}$ is a vector of the position parameters x, y, z, and $\mathbf{o}$ a vector of the orientation parameters Φ, Θ, Ψ. $\mathbf{f}_1(\mathbf{X})$ refers to the combined pose error function, considering the position and orientation error as separate objectives. The position error $f_1(\mathbf{p})$ and orientation error $f_1(\mathbf{o})$ use the Euclidean distance between the desired and the learned output [226].

Correlation Is All You Need

Correlation Coefficient (f_2): As an additional objective to enhance the learning process, a transformed version of the Spearman correlation coefficient was used as introduced in [299]. It describes the correlation between the produced and the desired output of a symbolic model. The main idea behind employing correlation next to the error is to evolve programs that are not yet numerically meaningful but already produce results that correlate with the desired output. Ideally, only small changes in these programs are necessary to

produce numerically accurate outputs. Moreover, a high negative correlation can contribute positively to the learning process, as only one mathematical operation is necessary to invert the results. Therefore, we used the absolute value of the correlation coefficient: $f_2 = 1 - |\rho|$. When using the angle error $f_1(\theta)$ as the first objective, ρ represents the correlation between the produced and the desired angle values. For the pose error $\mathbf{f}_1(\mathbf{X})$, three parameters for positioning and three for orientation needed to be considered. To this end, one correlation coefficient was calculated for each position and orientation. The final ρ was equal to the mean of the two absolute values of the position and orientation correlation coefficients [226].

Unit-Aware Genetic Programming

Dimension Penalty (f_3): Dealing with different physical units such as [rad] and [m] in a symbolic model is a challenging task, particularly given that trigonometric functions convert [rad] to dimensionless quantities. The third objective, f_3, is formulated to guide the algorithm to evolve programs which adhere to violate physical laws and match the target unit (i.e., [rad] for each joint angle of the manipulator). Different implementations of a dimension penalty in GP can be found in the literature [e.g., 128, 163, 175, 287] as described in Sec. 3.3.2. In the proposed approach, we compute the objectives f_1 and f_2 solely based on their numerical values without any unit feasibility check. For f_3, we traversed the GP tree using post-order traversal and checked each operation within an individual for unit matching. Penalties were aggregated throughout the individual, whereby each infeasible operation increased the penalty value by 1. Given that an operation was performed on incompatible units, the unit of the first input argument was handed over to the next operation, i.e., when an angle in [rad] and a length in [m] were added, this operation was penalized with a value of 1, and the unit [rad] was handed over to the next operation. Similarly, when an individual produces a unit that does not match the target unit, a value of 1 was added to the penalty variable. The objective function for the dimension penalty can be formulated as $f_3 = \text{dimPen}_{\text{op}} + \text{dimPen}_{\text{out}}$, where $dimPen_{op}$ represents the aggregated dimension penalties produced by non-physical operations within the individual and $dimPen_{out}$ is either 0 or 1, depending on whether the output unit is equal to the target unit [226].

4.3.2 Cooperative Coevolutionary GP for Inverse Kinematics

The Concept of Cooperative Coevolution

Cooperative coevolution is a concept in evolutionary algorithms that simultaneously evolves multiple so-called subpopulations to solve a problem. There are several variants of cooperation between the subpopulations. In some studies, equations evolved in different subpopulations are combined into one equation before evaluation [232]. In our approach, we kept the solutions separate, but evaluated them in a combined fitness measure as a collective evaluation. In this way, we accounted for the fact that all joints mutually influence each other and need to collaborate as a kinematic chain to achieve a desired pose. Here, one subpopulation per joint is evolved, while the other subpopulations are represented each by one representative individual. For the collective evaluation, we employed the FK equations to minimize the error between the desired pose and the learned pose in Cartesian space, which attained promising results in [262, 71]. This was the main idea behind the proposed IK-CCGP.

Cooperative Coevolution Within the Kinematic Chain

Figure 4.3 depicts one iteration of the IK-CCGP approach, as described in lines 6–15 in the complete Algorithm 3. The coevolutionary learning process is defined by multiple phases: While one subpopulation is evolved, the remaining subpopulations stand idle and are not changed. Global representatives for each subpopulation that is currently not under training, guarantee the collective evaluation using the FK equations. Next to the coevolutionary setting, we employed two important concepts in our algorithms that contributed to the learning process.

Two-Phase Training Procedure

In the first concept, we introduced a two phase training strategy. In the first training phase, Function `GP` (Algorithm 3, line 11) called a standard GP algo-

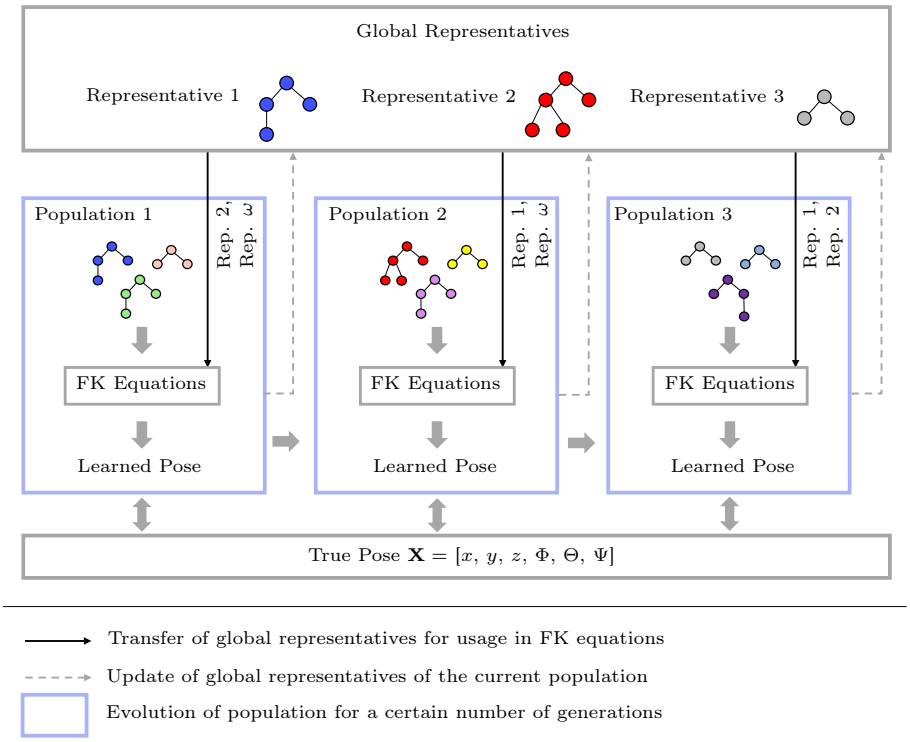


Figure 4.3: One iteration of the IK-CCGP approach [226].

rithm for k generations with crossover and mutation probabilities $p_c = p_m$ [299]. In the case of crossover, either one-point crossover or leaf-based one-point crossover was chosen at random. Mutation selected randomly between uniform mutation, node replacement, insertion mutation and shrink mutation. The second phase was a mutation-only phase for k generations to refine the current individuals in terms of slight changes in the primitives. This furthermore had the effect of preventing uncontrolled growth of the equations. Thus, Function `MutOnlyGP` (Algorithm 3, line 12) only selected from mutation operations, i.e., node replacement with a probability of 2/3 and shrink mutation with 1/3. In both Functions `GP` and `MutOnlyGP`, the archive A_j is updated according to the Pareto-dominance criterion [226].

Removal of Effects of Previous Joints from Input Data

For the second concept, we employed domain knowledge and provided information about the angles of the previous joints to the currently trained joint to steer the learning process towards a closed form solution. This idea arose from the fact that there exist multiple joint configurations for the same pose. Thus, even when all other joint values would be learned using the objective function $f_1(\theta)$, deviations in only one joint value could lead to an extremely inaccurate final pose. Inverse input transformation reverts the influence of all joints that are located before the currently learned joint in the kinematic chain. To revert the influence of a joint i, denoted by θ_i, from the target pose ${}^{i}\mathbf{X}_{\omega}$, consecutive rotations, denoted by Rot, and Translations, denoted by Trans, were executed. This resulted in the following transformation, using the four D-H parameters introduced previously [226].

$$ {}^{i+1}\mathbf{X}_{\omega} = t(\theta, i)\ {}^{i}\mathbf{X}_{\omega} = (\mathrm{Rot}_{\alpha_i})^{-1}(\mathrm{Trans}_{a_i})^{-1}(\mathrm{Trans}_{d_i})^{-1}(\mathrm{Rot}_{\theta_i})^{-1}\ {}^{i}\mathbf{X}_{\omega} \quad (4.3) $$

${}^{i+1}\mathbf{X}_{\omega}$ is the transformed target pose in the coordinate frame of joint $i+1$, which is used in the training of joint $i+1$. The offset parameters were constant for a given joint. Algorithm 2 describes this consecutive transformation for all joints of the kinematic chain: the training data was parsed through the representative of θ_1 and transformed the training data by the resulting angle values using Eq. 4.3. The transformed pose was used as input data for the individual of θ_2, and the process repeated iteratively until θ_{ω} is reached.

Algorithm 2: IK-CCGP Evaluation Procedure for One Individual [226]

input : Individual ind_j, Training Data $\mathbf{X}$, Set of Representatives r, Joint Index j
output: Objective values

1 $inds \leftarrow [r_1, \ldots, r_{j-1}, ind_j, r_{j+1}, \ldots, r_\omega]$
2 ${}^1\mathbf{X}_\omega \leftarrow \mathbf{X}$
3 **for** $i \leftarrow 1$ **to** ω **do**
4 $\quad \theta_i \leftarrow$ parse ${}^i\mathbf{X}_\omega$ through $inds_i$
5 $\quad {}^{i+1}\mathbf{X}_\omega \leftarrow t(\theta_i, i)\ {}^i\mathbf{X}_\omega$ `// as outlined in Eq. 4.3`
6 **end**
7 $\hat{\mathbf{X}} \leftarrow g(\boldsymbol{\theta})$ `// use FK equations`
8 $\mathbf{f} \leftarrow [f_1, f_2, f_3]^T$
9 **return** $\mathbf{f}$

Algorithm 3: IK-CCGP Algorithm [226]

input : Training data $\mathbf{X}$, number of subpopulations ω, subpopulation size N, number of generations k, number of iterations l
output : Set of archives A_j, $j = 1, \ldots, \omega$

1 **for** $j \leftarrow 1$ **to** ω **do**
2 $\quad P_j \leftarrow$ randomly initialize subpopulation with size N
3 $\quad r_j \leftarrow$ select global representative from P_j
4 $\quad A_j \leftarrow$ initialize empty archive
5 **end**
6 **for** l iterations **do**
7 $\quad$ **for** $j \leftarrow 1$ **to** ω **do**
8 $\quad\quad$ **for** $i \leftarrow 1$ **to** N **do**
9 $\quad\quad\quad$ evaluate ($P_{j,i}$, $\mathbf{X}$, r, j) `// evaluation in Algorithm 2`
10 $\quad\quad$ **end**
11 $\quad\quad$ $P_j, A_j \leftarrow$ `GP`(k, $\mathbf{X}$, P_j, A_j) `// evaluation in Algorithm 2`
12 $\quad\quad$ $P_j, A_j \leftarrow$ `MutOnlyGP`(k, $\mathbf{X}$, P_j, A_j) `// evaluation in Algorithm 2`
13 $\quad\quad$ $r_j \leftarrow$ update global representative using A_j
14 $\quad$ **end**
15 **end**
16 **return** $\bigcup_{j=1}^{\omega} A_j$

The Overall Algorithm

These two concepts are included in Algorithm 3. The algorithm requires a set of training data $\mathbf{X}$, the number of joints or subpopulations ω, and the subpopulation size N. Additional input parameters are the number of generations per training phase k and the number of training iterations l. Initially, one representative is randomly selected from each of the initial subpopulations (lines 1-4). In the main loop (lines 6-15), the subpopulations are evolved using the above two-phase training strategy. The evaluation procedure for an individual in `GP` and `MutOnlyGP` calls Algorithm 2. In Algorithm 2, first (line 1) a list of representatives and the individual to be evaluated is created in the order of the joints in the kinematic chain. For example, when joint θ_2 is currently trained and the kinematic chain consists of $\omega = 4$ joints, the list contains the elements $[r_1, ind_2, r_3, r_4]$, denoted by $inds$. This list is used to perform the inverse transformation of the training data in lines 3-6, as introduced previously. The resulting joint angles $\boldsymbol{\theta}$ are fed into the FK equations to compute the resulting pose $\hat{\mathbf{X}}$ (Algorithm 2, line 7). This pose can then be compared to the true pose $\mathbf{X}$ using the objective function $f_1(\mathbf{X}, \hat{\mathbf{X}})$. Depending on the application, additional objectives can be computed. After k generations of the two-phase training, one representative from the subpopulation is selected, which is used to determine the joint value for this subpopulation during the evolution of the remaining subpopulations. After l iterations of the main loop, the algorithm terminates and returns a Pareto-dominance-based archive of cooperating solutions. Each solution is a non-dominated individual of a subpopulation, together with the representatives of the other subpopulations that were used during the

evaluation and calculation of the objective values. In general, the proposed approach can be applied to either the entire kinematic chain of a robot, or a kinematic unit within the kinematic chain that consists of multiple consecutive joints [226].

4.4 Experiment Setup

Specifications of the KUKA youBot

To evaluate the proposed approach, we applied the proposed algorithm to develop an inverse kinematics model $\theta = g^{-1}(\mathbf{X})$ of the *KUKA youBot* with 5 DOF. The function g has the domain $[-0.4\ m; 0.4\ m]^3 \times [-\pi\ rad; \pi\ rad]^3$, which corresponds to the reachable area of the *KUKA youBot*. The co-domain is given by the possible range of the joints angles. This manipulator comes with a special setup: The first joint plays an important role as it determines the plane, in which the target position is located. This plane can be defined by two configurations of the base joint, which are $\pm\pi$ apart from each other. When the robot bends over itself, the first joint is directed towards the opposite side of the target position. Hence, an angle error of $f_1(\theta) = \pm\pi$ rad can be produced when an equation learns an angle that also defines the plane but is shifted by π. We considered two different variants of input angles of θ_1 for the training of joints > 1: First, the true θ_1 angle including bend-overs (this requires a bend-over handling strategy); second, the projected angle resulting from the projection of the target pose onto the x-y-plane, which excludes bend-overs. The projected angle can be calculated with the simple equation

$$\theta_1 = \frac{169}{180}\pi - \arctan2(y,\ (x - 0.024)) \tag{4.4}$$

where $\alpha_1 = \frac{169}{180}\pi$ and $d_1 = 0.024$ are known offset parameters of the first joint. The major goals of the experiments were to find out which combination of objective functions performed best, and how the proposed IK-CCGP performed on a kinematic unit of two joints compared to a kinematic chain of three consecutive joints. Furthermore, we intended to measure to what extent the outcome is affected by transforming the input data for joints 2 and 3 by the *projected* angle of θ_1 compared to transforming by the *ground truth* angle of θ_1.

4.4.1 Inverse Kinematics Benchmark

Data Generation

To generate the training data for the IK problem, we implemented an FK algorithm according to the D-H convention for the *youBot*. For each of the five joints, 20 discrete positions were selected, evenly distributed within the movement range of each joint. For all 20^5 combinations of joint values, the end-effector pose in Cartesian space is computed using the FK equations. Each training sample incorporates the pose parameters $x,\ y,\ z,\ \Phi,\ \Theta,\ \Psi$ and corresponding joint angles $\theta_1,\ \theta_2,\ \theta_3,\ \theta_4,\ \theta_5$ as features. A robot simulation environment performed a feasibility check on each data instance, i.e., whether the configuration led to a feasible pose or caused self-collision of the arm or collision with the ground. All infeasible samples were removed from the training data set, leaving 949,593 feasible data samples within the robot workspace that could be employed for training the IK-CCGP algorithm. We extracted a representative training dataset with 10,000 samples evenly distributed over the entire workspace. For the final evaluation of the found solutions, we employed 20,000 data samples randomly drawn from the workspace.

Data Transformation

As described in the previous section, we transformed the input data for joints > 1 according to Eq. 4.3 to remove the influence of previous joints. Fig. 4.4 shows an exemplary distribution of the training data, where Fig. 4.4a displays input data for training of θ_1, and Fig. 4.4b the transformed input data for θ_2. It has a planar shape since the influence of the first joint is removed. In this way, the target position depended solely on two instead of three variables.

Batch Training

In order to reduce the computational cost, the algorithms were trained on

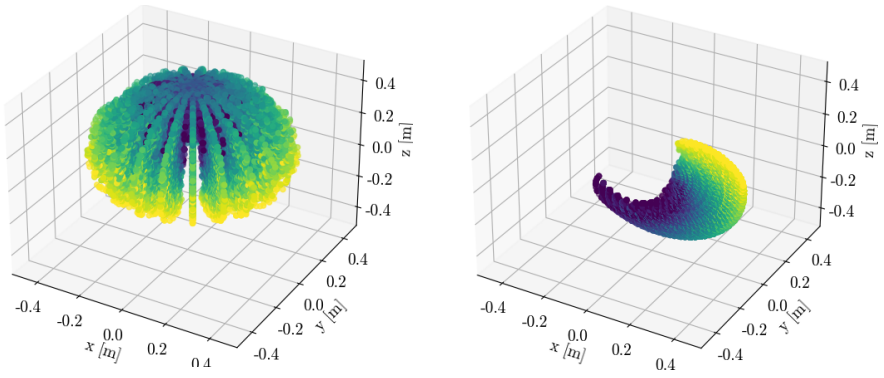


Figure 4.4: Data samples in the workspace of the manipulator. Colors represent magnitude of angle values of θ_2 [226].

(a) Input data to compute θ_1. No transformation.

(b) Input data to train θ_2. Influence of θ_1 is removed by inverse transformation.

batches of data with a batch size of 200. The data samples per batch were assigned in advance to ensure a fair comparison between the algorithmic variants.

4.4.2 Algorithm Variants

Preliminary Experiments

We ran experiments in two stages: in preliminary experiments, we identified the best combination of objective functions for the IK problem by performing the training of θ_2 on the angle error $f_1(\theta)$ and the additional objectives f_2 and f_3. We selected θ_2 as a use case since a simple equation was not known beforehand, besides for the first joint, where we either required a bend-over handling technique for the ground truth angles or used the simple formula for projected angles. The best combination of objective functions was employed in the IK-CCGP experiments, where we challenged our proposed IK-CCGP approach.

IK-CCGP Experiments

To compare the performance of our approach on different numbers of joints involved in the coevolution, we tested two scenarios: First, we trained the first three joints of the kinematic chain of the *KUKA youBot*, which included the base joint θ_2 and θ_1 and the first kinematic unit consisting of θ_2 and θ_3. Second, to understand the influence of the different θ_1 input variants *ground truth* and *projected*, we took the angle of the first joint as given and evaluated how the proposed approach reacts to the two different input types. To get an impression about the quality of our proposed approaches, we only considered the kinematic unit consisting of θ_2 and θ_3 in this scenario. Since the orientation of a pose is mostly determined by the last two joints, we optimized and evaluated the outcome of the experiments only on the position error $f_1(\mathbf{p})$ and excluded the orientation error from consideration. This made a total of three experiment instances for IK-CCGP experiments: IK-CCGP-3 that applied our approach on the first three consecutive joints of the kinematic chain, IK-CCGP-2G and IK-CCGP-2P using ground truth (G) and projected (P) θ_1 input data respectively applied to the kinematic unit of θ_2 and θ_3.

Choice of Hyperparameters

All experiments used the same function set $\mathcal{F} = \{+, -, \cdot, /, \cos(\circ), \sin(\circ), \tan(\circ), \arccos(\circ), \arcsin(\circ), \arctan2(\circ, \circ), -\circ, \circ^2, \sqrt{\circ}, \circ \bmod 2\pi\}$, where $\circ$ represented the input of unary operators. The terminal set $\mathcal{T}$ varied for each experiment and consisted of the six parameters that define the (ground truth or projected) target pose and additionally a set of constants $\mathcal{C}$ containing the offset parameters d_i and α_i of the joints that are currently trained: $\mathcal{T} = \{x, y, z, \Phi, \Theta, \Psi\} + \mathcal{C}$. For each algorithmic variant, 31 independent runs were performed. The parameters were set as follows: In the single-objective optimization, we used tournament selection with a tournament size of 3. We used NSGA-II algorithm for multi-objective optimization [65]. The population size was $\mu = \lambda = 1500$ for all experiments. We used crossover and mutation

Table 4.1: Overview of domain knowledge integrated as bias into the GP algorithms proposed in this chapter.

Bias Type	Bias
Observational	Transformation of data and reversion of influence of previous joints before evaluation (see Fig.4.4 and Eq. 4.3)
Observational	Consideration of the role of the first joint with two variants in experiments IK-CCGP-2G and IK-CCGP-2P (see Sec. 4.4.2)
Inductive	Cooperative coevolution to represent the collaboration of joints within the kinematic chain to reach a target pose (see Algorithm 3)
Learning	Dimension penalty for equations with unit violations (see Sec. 4.3.1)
Learning	Problem-specific pose error objective including FK equations (see Sec. 4.3.1)

Table 4.2: Pairwise statistical comparison of the best individuals using different objective functions for θ_2 ($+$ / $-$ / $\sim$: row significantly better/ worse / no difference compared to column) [221]

	f_1	$f_1 f_2$	$f_1 f_3$	$f_1 f_2 f_3$
f_1		$-$	$+$	$\sim$
$f_1 f_2$	$+$		$+$	$+$
$f_1 f_3$	$-$	$-$		$-$
$f_1 f_2 f_3$	$\sim$	$-$	$+$	

probabilities of 0.5, except for the mutation-only phase, with a mutation probability of 1.0. The leaf-biased crossover selected a leaf with a probability of 0.8. The maximum depth of a solution tree was set to 20 with a maximum of 30 nodes. The preliminary experiments executed ten algorithmic iterations, one of which consists of ten generations of crossover and mutation followed by ten generations of mutation-only, amounting to 200 generations in total. As the IK-CCGP experiments intended to solve a more complex problem, and coevolution requires more time for mutual adjustment between the subpopulations, $l = 20$ iterations of the IK-CCGP algorithm were performed, where each subpopulation was trained for $k = 10$ generations in each phase of the two-phase training. This resulted in a total of 800 generations for IK-CCGP-2G and IK-CCGP-2P and 1200 for the three joint experiment IK-CCGP-3. All algorithms were implemented using the `DEAP`-framework version 1.3.1 [90] and the `pint` package version 0.16.1.Tab. 4.1 summarizes how the available domain knowledge was integrated as bias into the algorithm.

4.5 Results and Analysis of Preliminary Experiments

Selection from Final Population and Statistical Test

In the preliminary experiments, we tested the performance of the combinations of objectives $\mathbf{f} = [f_1(\theta)]$, $\mathbf{f} = [f_1(\theta), f_2]^T$, $\mathbf{f} = [f_1(\theta), f_3]^T$ and $\mathbf{f} = [f_1(\theta), f_2, f_3]^T$. We always included the error function $f_1(\theta)$, since it is the main objective we wanted to minimize. The individual with the lowest RMSE according to $f_1(\theta)$ on the evaluation dataset was selected as the best solution for each of the 31 runs. Thus, for each experiment variant, 31 RMSE values were employed to determine the quality of the learned solutions. We conducted the pairwise Wilcoxon–Mann–Whitney rank sum test with a level of significance $\alpha = 0.05$ to compare the performance of different combinations of objective functions [226].

The error and correlation objectives performed the best.

The results of the statistical test in Tab. 4.2 indicated that the multi-objective variant $\mathbf{f} = [f_1(\theta), f_2]^T$ was superior to the single-objective variant $\mathbf{f} = [f_1(\theta)]$. This implies that the correlation as an additional objective enhances the quality of the results. The combination $\mathbf{f} = [f_1(\theta), f_3]^T$ yielded the worst results in terms of RMSE. The three-objective variant, which also incorporated corre-

lation, could partially compensate for the drawbacks of the dimension penalty objective. Nonetheless, it was outperformed by the two-objective variant using $\mathbf{f} = [f_1(\theta), f_2]^T$ [226].

4.6 Results and Analysis of IK-CCGP Experiments

Based on the preliminary experiments, all IK-CCGP experiments were conducted using the combination of objectives $\mathbf{f} = [f_1(p), f_2]^T$.

4.6.1 Convergence Behavior

Fig. 4.5a displays the convergence behavior of the position error for the IK-CCGP experiments. This data was deduced from the training process, which started 400 generations later for IK-CCGP-2G and IK-CCGP-2P, as they did not include the training for θ_1. The minimum error for each of the 31 experiment runs is averaged at different times during the training for each experiment type.

IK-CCGP-3 showed unsteady convergence behavior.

From this plot, one may infer that experiment variant IK-CCGP-3 performed the worst. Especially the zigzag pattern of the gray curve indicates that the learning process did not follow a continuous improvement, but rather oscillated around a slowly declining curve. This behavior can be explained by the fact that θ_1 was an additional variable in IK-CCGP-3. Since the first joint defines the plane in which the final position lies within the workspace, errors in this joint can lead to large deviations in the final position [226].

IK-CCGP-2P and IK-CCGP-2G had both similar convergence behavior, resulting in a position error of a few centimeters.

It can be observed that it is of great advantage when the angle value of the first joint is already known, as was the case in IK-CCGP-2G and IK-CCGP-2P. In this way, the algorithm only operates towards finding the correct position on the predefined plane, which shrinks the search space tremendously. IK-CCGP-2G and IK-CCGP-2P followed more the expected fitness development, with poor results at the beginning, which rapidly improved in the first quarter of the evolution process and converged towards the end. In general, both experiments produced position errors in the magnitude of a few centimeters [226].

4.6.2 Distribution of Errors

The best solution by IK-CCGP-2P performed the best, with a median RMSE of 0.0213m.

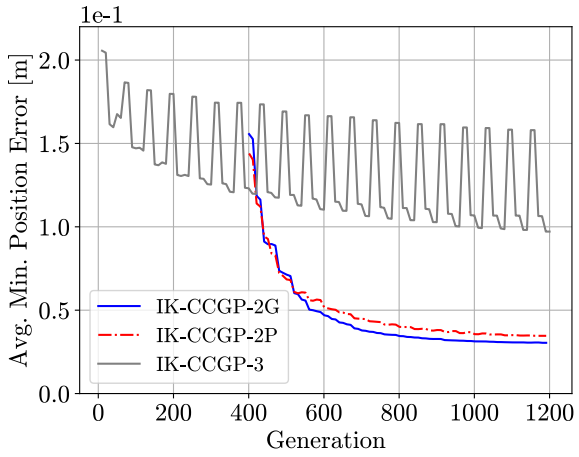
For further analyses, we selected the solution with the smallest position RMSE among the 31 experiment runs for each of the experiment variants. Fig. 4.5b displays the distribution of position errors on the evaluation dataset for these best solutions. It became apparent that the best solution of IK-CCGP-3 performed the worst of all other solutions, with a median error of 0.0671m. The other variants ranged between 2cm and 3cm of median error. No considerable difference between using the ground truth or projected θ_1 angles as input data could be observed. The overall best solution, with an RMSE of 0.0343m, a maximum absolute error of 0.1712m, and a median error of 0.0213m, was obtained by experiment variant IK-CCGP-2P. In general, the two joint variants yielded solutions with zero errors in certain areas of the workspace. However, the presence of fliers in the box plot, indicated by errors above the third quartile plus 1.5 times the interquartile range, suggests that the model had difficulty predicting some data points [226].

Large errors in the prediction of IK-CCGP-2P were mostly located at the center top of the workspace.

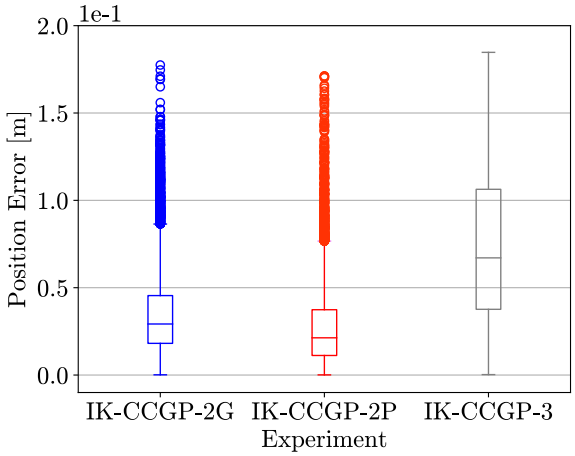
We analyzed this distribution and the percentage of large error values of more than 0.1m. Again, IK-CCGP-3 with 28.3% had the largest percentage of error. The approach with the smallest percentage of large errors was IK-CCGP-2P with 0.8%, compared to IK-CCGP-2G with 2.1%. Fig. 4.6 gives additional clues about the distribution of large errors within the workspace. For all data samples which caused large position errors, the original positions are plotted to identify problematic regions. The IK-CCGP-2P algorithm mainly had problems finding joint values for positions at the top of the reachable area and very few outliers

Figure 4.5: Results of the IK-CCGP experiments [226].

(a) Convergence behavior of the three experiment variants. The variants learning only two joints start later, as they skip θ_1.



(b) Error distribution on the evaluation dataset with 20,000 samples using the best solution of each experiment variant.



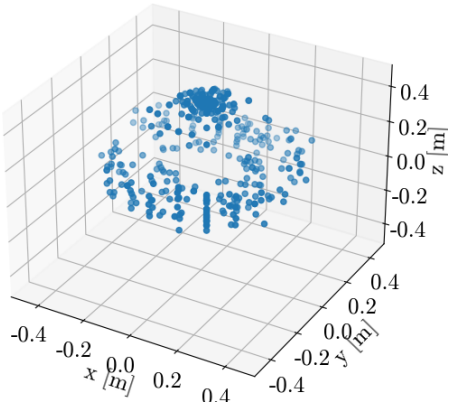
in the remaining workspace. Additionally, IK-CCGP-2G produced large errors at the bottom of the workspace and the edge of the reachable area.

Multimodality inherent to the IK problem might have been the reason for less consistent solutions obtained with IK-CCGP-2G.

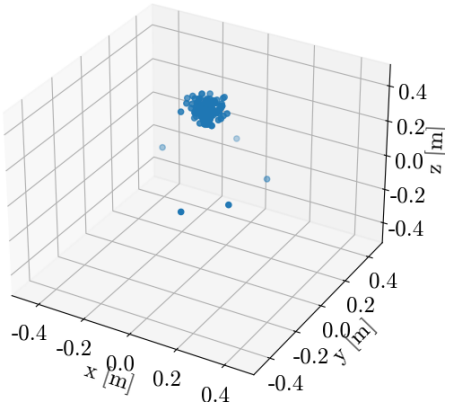
The algorithm found more consistent solutions throughout the workspace when the projected θ_1 angles were given as inputs, i.e., IK-CCGP-2P, compared to the ground truth angles in IK-CCGP-2G. An explanation for this observation is that IK-CCGP-2G was trained on the ground truth data of θ_1. In this way, the algorithm might have received input values of θ_1 that were $\pm\pi$ apart but arrived at positions that were very close to each other, once with a bend over and once without. The multimodality inherent to the IK problem and reflected in the input data of IK-CCGP-2G is likely to lead to jumps in the convergence process or inconsistencies in the predictions. Conversely, IK-CCGP-2P received very consistent input angles of θ_1, i.e., positions that were very close to each other also originated from the same θ_1 angle. The multimodality was thereby partially removed from the training data. The use of continuous input enabled the algorithm to learn a consistent model for most positions within the workspace. Two possible explanations for the cluster of large errors above the center of the x-y-plane with z-values between 0.33m and 0.47m arose from this assumption: First, these positions could only be reached by a bend over due to the kinematic configuration of the robot, and second, the algorithm had issues

Figure 4.6: Distribution of large position errors > 0.1 m evaluated on 20,000 data samples [226].

(a) IK-CCGP-2G.



(b) IK-CCGP-2P, z values of cluster in the center range from $0.33\,m$ to $0.47\,m$.



CCGP-	θ_i	Individual	RMSE [m]	> 0.1m
-2G	2	$\frac{\tan\left(\tan\left(\left(\arcsin\left(\sqrt{z^*}\right)\right)^2+x^*\right)\right)}{\arccos\left(\tan\left(\arcsin\left(\sqrt{x^*}\right)\right)\right)} + \\ + \arctan\left(\tan\left(x^*\right), -\left(z^* - \arctan\left(d_2, \left(z^* + \arcsin\left(\cos\left(\Theta^*\right) \cdot x^*\right)\right)\right)\right)\right)$	0.0415	2.1%
	3	$-d_3 - \sin\left(x^*\right)$ - \\ - $\left(\tan\left(x^* + x^* + d_3\right) \cdot \arccos\left(\arcsin\left(\arctan\left(\sin\left(\Psi^*\right), \alpha_3\right) \cdot \cos\left(\arctan\left(\alpha_2 \cdot x^*\right), \frac{\alpha_3}{\Phi^*}\right)\right)\right)\right)^2$		
-2P	2	$z^* + \alpha_2 + \left(\arctan\left(z^*, x^*\right) + z^* \cdot \cos\left(\arccos\left(\alpha_2 + \Psi^* - \arcsin\left(-\left(\tan\left(\Theta^*\right)\right)\right)\right)\right) + \right. \\ \left. + \tan\left(\tan\left(\sqrt{\tan\left(d_3 \cdot \Phi^*\right)} - d_2\right)\right)\right.$	0.0343	0.8%
	3	$mod2\pi\left(\left(x^* + \alpha_2 + \alpha_3 - \frac{\left(\sin\left(d_2\right)+d_3\right) \cdot \sin\left(\Psi^*+d_2\right)}{\Phi^*}\right)^2\right)$		
-3	1	$mod2\pi\left(\tan\left(\alpha_1\right)\right) - \arctan\left(y, \left(\left(\sin\left(-d_1\right) \cdot \alpha_3\right) + x\right)\right)$	0.0856	28.4%
	2	$\frac{d_2}{d_3} - \sin\left(\alpha_3 \cdot z^*\right) - \left(y^*\right)^2 - d_3$		
	3	$\left(\arccos\left(\left(mod2\pi\left(x^*\right) - d_1\right)\right)\right)^2 + \\ + \left(\arctan\left(\left(x^*\right)^2, d_1 - \left(y^* + d_1\right)\right) + \left(\sqrt{d_3} \cdot \arctan\left(\arctan\left(x^*, \left(\Phi^* + z^*\right)\right), \left(y^* + d_1\right)\right)\right)\right)$		

Table 4.3: Best individuals of each variant of the IK-CCGP experiments, evaluated on a test dataset with samples from the entire workspace. The last column displays the percentage of samples with a prediction error above 0.1 m [221]

learning joint angles that completely extended the arm to the top.

4.6.3 Resulting Equations

Frequent nesting of trigonometric functions was observed in the resulting equations.

It is crucial to have control and to be aware of the potential reactions of the manipulator in all conceivable scenarios, especially in safety-relevant situations, where other machines or humans may collaborate with the robot. The main motivation to employ GP methods for solving the IK problem is the resulting output of human-readable equations, which increase interpretability and mathematical control. To gain more profound insights into the evolved models, Tab. 4.3 displays the best combination of equations evolved with GP using the three tested algorithms. The transformed variables are marked with an asterisk symbol. First, it becomes apparent that the trigonometric functions provided in the function set played an important role in the prediction of joint angles. Almost all equations used nested trigonometric functions, such as evolved by IK-CCGP-2G employing the term $\tan\left(\tan\left(\left(\arcsin\left(\sqrt{x^*}\right)\right)^2 + x^*\right)\right)$. However, the nesting of trigonometric functions and the use of higher-order combinations, such as double tangent and arcsin terms, can result in expressions that are both complex and numerically unstable. Such combinations of operators are relatively uncommon in engineering sciences, which makes it challenging to interpret them from a physical or engineering perspective, as they do not align with established physical principles.

Most equations used only the relevant features, but exhibited dimensional inconsistencies.

From a dimensional analysis standpoint, it is evident that almost all equations contained some degree of inconsistency in their dimensional representation. For example, the equation for θ_3 developed by IK-CCGP-2P included the term

$(\Psi^* + d_2)$. Interestingly, the equations developed by IK-CCGP-2G and IK-CCGP-2P for θ_2 used only the position coordinates x^* and z^*, and omitted y^*, which was a constant value after the pose transformation, as shown in Fig. 4.4b. Moreover, the subsequent joint θ_3 only used the transformed x^* as a dependent position variable, as well as two Euler angles for orientation. Thus, the models only used the relevant and informative features after the pose transformation.

The equation for θ_1 learned by IK-CCGP-3 was structurally similar to Eq. 4.4, with a few differences.

The same behavior cannot be observed for the equations developed by IK-CCGP-3, which was additionally tasked with learning a model for θ_1. Structurally, the model for θ_1 shared similarities with the equation for the projected joint angle in Eq. 4.4. Once again, the multimodal nature of the IK problem might have increased the difficulty for the algorithm to converge on a single, stable solution across all input scenarios. Errors in θ_1 led to an accumulation of errors for θ_2 and θ_3, resulting in the highest percentage of large errors among the tested algorithms.

4.7 Discussion and Limitations

Comparison of Error Rate with ANNs

Our main reason for preferring GP over ANNs is the explainability of the produced solutions. However, the most important criterion to assess and compare IK models is the error rate. Most IK models developed with ANNs are evaluated using continuous trajectory tracking: once the start position of the trajectory is found, the subsequent positions are close to the current position [262, 257]. Our evaluation method used 20,000 independent positions instead of a continuous trajectory, which makes it difficult to compare our accuracy to ANNs. Multiple publications report position errors of ANNs in the range of millimeters [285], below 1 percent in the x, y, and z position components [5], and up to a few centimeters in some cases [257]. The best solution developed by IK-CCGP has a median error of 2.13cm on the position of the third joint, without considering the orientation component. However, the comparably large median error of 0.0671m and RMSE of 0.0856m when three consecutive joints are trained, makes the developed equations too inaccurate to be used in real-world applications. Yet, it became apparent that approximate symbolic models with a reasonable level of complexity existed, which were capable of capturing the variability in the output to a certain extent. Since the existing GP approaches for the IK problem use an angle error in [rad], a fair comparison between our approach using position errors in [m] and the existing GP approaches is not possible [39, 40].

Limitations of the Proposed Approach

The proposed IK-CCGP approach had several limitations that need to be considered. Firstly, while the addition of a dimension penalty was intended to ensure dimensional consistency, it led to a deterioration in the RMSE when evolving equations for θ_2, for which no ground truth is known. However, without the dimension penalty, the resulting equations were not conformal with physical units, which is a fundamental requirement from both science and engineering perspectives. Furthermore, the approach relied solely on constants provided in the terminal set, without applying any additional constant fitting. Another challenge encountered was related to the co-evolutionary strategy. The algorithm struggled to effectively determine the contributions of each joint within the kinematic chain to achieve the desired final pose, which became clear when a kinematic unit of three joints was trained. This indicates the need for better mechanisms to guide the search process in such complex environments. As described in Sec. 4.2, alternative methods have been proposed that approximate manipulator behavior using GNNs. These methods furthermore provide a strong inductive bias for GP, suggesting that there may be more suitable approaches for this problem domain compared to co-evolution.

4.8 Summary

In this chapter, the IK-CCGP approach to solve the inverse kinematics problem using genetic programming was proposed. The main goal was to overcome the explainability gap of ANNs. A cooperative coevolutionary setting using a two-phase training strategy was introduced. To include information about the joint angles of previous joints, we employed inverse transformation of training data. We furthermore introduced different objective functions, one of which employed the FK equations to compute the pose from the learned joint angles. Overall, domain knowledge about robot manipulators and their behavior was integrated as observational, inductive, and learning biases. The proposed approach was evaluated by tasking it with the development of an IK model for the 5 DOF *KUKA youBot* manipulator. In preliminary experiments, we identified the combination of objectives error and correlation, $\mathbf{f} = [f_1, f_2]^T$, as fitting for our purpose. In the IK-CCGP experiments, we tested the IK-CCGP approach in three scenarios. The experiments for learning three consecutive joints of the kinematic chain performed worst. Experiments that learned a kinematic unit of two joints generated promising results in the magnitude of a few centimeters of position error. However, there were still some limitations regarding dimensional consistency, the use of constants, and the representation of the kinematic chain through co-evolution. This work motivated improvements at various levels, which will be further elaborated on later in this thesis.

5 Simulation of Particle-Laden Flows

Introduction

This chapter introduces another real-world problem from the field of fluid mechanics to serve as a benchmark for evaluating existing methods in GP when applied to complex domain-specific problems. We use domain knowledge to enhance out-of-the-box GP algorithms to tackle a problem with unknown analytical solution. The insights gained from the presented approach provide a foundation for the advancements introduced later in this thesis.

This chapter is largely based on the author's publication [222]. The author thanks Prof. Dr.-Ing. Berend van Wachem and Hani Elmestikawy for providing datasets, domain knowledge, and expert insights about fluid mechanics and particle-laden flows, and for critically reviewing draft versions of this chapter.

5.1 Methods to Simulate Particle-Laden Flows

Particle-laden flows are omnipresent in our daily lives.

Particle-laden flows are complex fluid systems in which solid particles are suspended and transported by the surrounding fluid. They can be encountered in many natural and engineering processes, such as the flow of blood cells in blood plasma, flow of sediments in rivers, or the fluidization of biomass particles in industrial furnaces. In other words, they are omnipresent in our daily lives. Yet, accurately simulating and predicting the behavior of such flows remains a challenging task because of the multiscale character of the interactions within the fluid, between the particles and the fluid, as well as between the particles themselves.

Stokes Flow and Governing Equations

This thesis examines particle-laden flows with perfectly spherical particles in the Stokes regime, where viscous forces dominate over inertial forces. The Stokes regime is characterized by a low Reynolds number $\mathrm{Re} \to 0$. The Stokes equations [263] can be considered a simplification of the general Navier-Stokes equations at low Reynolds numbers. They describe the motion of a viscous incompressible fluid, with the assumption that inertial and time-dependent effects can be neglected. The equations are given as

$$\nabla \cdot \mathbf{u} = 0 \tag{5.1}$$

$$-\nabla p + \mu \nabla^2 \mathbf{u} = -\mathbf{F} \tag{5.2}$$

where $\mathbf{u}$ is the velocity field of the fluid, ∇^2 is the Laplace operator, p is the pressure field, μ is the fluid viscosity, and $\mathbf{F}$ is the force per unit volume. $\nabla \cdot \mathbf{u} = 0$ is the incompressibility condition, which ensures mass conservation.

Stokes Flow Past a Single Spherical Particle

For a single rigid sphere with radius a in Stokes flow, subject to a far field flow with the velocity $\mathbf{u}_\infty$ and $||\mathbf{u}_\infty|| = u_\infty$, the Reynolds number satisfies

$$\mathrm{Re} = \frac{2a\rho u_\infty}{\mu} \ll 1. \tag{5.3}$$

For a single rigid spherical particle, the Stokes equations [263] can be expressed as:

$$-\nabla p + \mu \nabla^2 \mathbf{u} = 0 \tag{5.4}$$

with boundary conditions:

$$\mathbf{u} = 0 \quad \text{at } r = a, \tag{5.5}$$

$$\mathbf{u} \to \mathbf{u}_\infty \quad \text{and } p \to 0 \quad \text{as } r \to \infty. \tag{5.6}$$

As indicated by the boundary condition in Eq. 5.5, the fluid velocity at the surface of the particle is equal to 0, as the particle is stationary. Eq. 5.6 shows that the velocity disturbance introduced by the particle, along with the pressure field, decreases over distance from that particle. Thus, the fluid velocity approaches the far field velocity as the distance from the particle r goes to infinity [102].

In the Stokes regime, an analytical solution for the flow past a single spherical particle can be obtained. The stream function for the Stokes flow around a spherical particle in spherical coordinates with the origin at the center of the sphere is given by:

$$\psi(r, \theta) = u_\infty \sin^2\theta \left(\frac{r^2}{2} + \frac{a^3}{4r} - \frac{3ar}{4} \right), \tag{5.7}$$

resulting in the following equations that describe the velocity field [263]:

$$u_r(r, \theta) = \frac{1}{r^2 \sin\theta} \frac{\partial \psi}{\partial \theta} = u_\infty \cos\theta \left(1 + \frac{a^3}{2r^3} - \frac{3a^2}{r} \right) \tag{5.8}$$

$$u_\theta(r, \theta) = -\frac{1}{r \sin\theta} \frac{\partial \psi}{\partial r} = -u_\infty \sin\theta \left(1 - \frac{a^3}{4r^3} - \frac{3a}{4r} \right). \tag{5.9}$$

The resulting fluid forces $\mathbf{F}$ on a single sphere in Stokes flow can then be derived as

$$\mathbf{F} = 6\pi\mu a \mathbf{u}_\infty. \tag{5.10}$$

This equation shows that the forces are linearly proportional to the particle velocity $\mathbf{u}_\infty$ and the viscosity μ. Accurately predicting the forces acting on particles is essential for reliable simulations of particles suspended in a fluid.

Stokes Flow Past Multiple Spherical Particles

In Stokes flow, the behavior of a single spherical particle in a viscous fluid is well understood and can be described analytically. However, most practical scenarios involve numerous particles, where these principles are not applicable anymore. In such cases, a closed-form solution of the Stokes equations is generally not available, as the presence of multiple particles introduces new complexities, such as particle interactions and disturbances in the flow field introduced by surrounding particles. These factors lead to non-linearities and require more sophisticated approaches, such as numerical simulations or approximate models, to capture the full dynamics of many-particle systems in viscous flows.

The Stokes flow involving multiple particles is, next to the Reynolds number, also characterized by the volume fraction ϕ, which quantifies the fraction of the volume that is occupied by particles. In Stokes flow, the Reynolds number is typically very small ($\mathrm{Re} \ll 1$), and the volume fraction becomes more critical in determining the overall flow behavior.

Simulation of Stokes Flow Past Multiple Spherical Particles

Multiple approaches to simulate the behavior of particle-laden flows and the

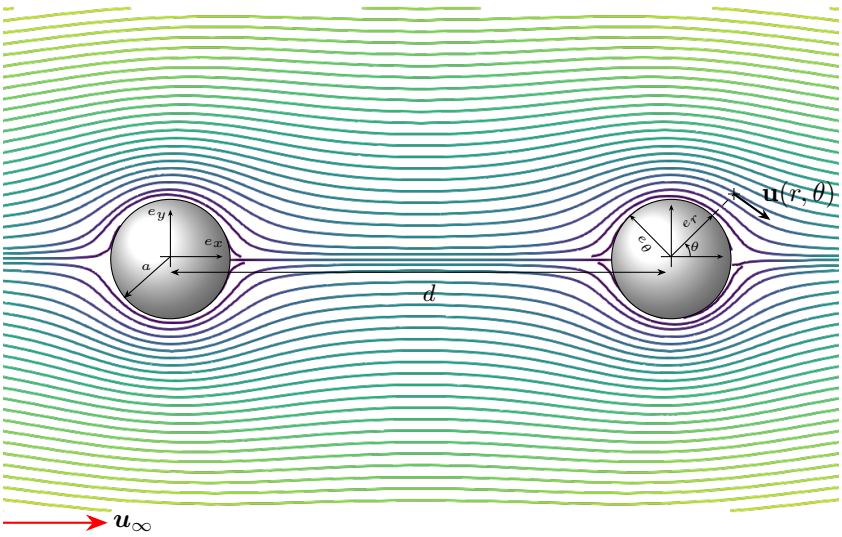


Figure 5.1: Streamlines of the Stokes flow around two spheres with radius a, separated by a distance d [222].

respective velocity fields are available in the literature. *Particle-resolved direct numerical simulation*s (PR-DNSs) fully resolve the hydrodynamic forces acting on individual particles and avoid the use of empirical closure models, delivering results with high fidelity. However, due to its computational costs, this approach becomes infeasible for complex particle arrangements as well as higher Reynolds numbers or densely packed systems with higher volume fractions [15]. The method of *Regularized Stokeslets* [56] is an alternative approach prevalent in simulations of particle-laden flows at low Reynolds numbers. Instead of solving the Stokes equations around each particle directly, the method uses force markers to apply force at discrete points smoothed by a regularization kernel, which represent the influence of the particles on the surrounding fluid. Each marker generates a velocity field, and the velocity at any point in the fluid is the sum of the velocity fields generated by all the force markers. Since the Stokes equations are linear, the total flow field generated by multiple force markers is approximated by superposing the contributions of individual Stokeslets. This method is only suitable for the Stokes regime, where the physics of the flow can be described by the linear Stokes equations.

The *volume filtered Euler Lagrange* (VFEL) approach [36] is a method to simulate the behavior of systems with many particles, as they frequently appear in practice, balancing accuracy and computational cost. It is based on the Euler-Lagrange framework, where the Eulerian part describes the fluid phase, and the Lagrangian part tracks the particles. VFEL captures the general behavior of the fluid-particle system at the meso-scale in the magnitude of multiple particle diameters, but does not resolve individual particle-fluid interactions at fine scales. As volume filtering results in unresolved sub-grid scales, closure models are necessary to estimate the forces and interactions that cannot be directly simulated at the meso-scale. Closure is required on the hydrodynamic force $\mathbf{F}$ with its components drag F_D and lift F_L, the torque $\mathbf{T}$, the sub-filter stress $\mathbf{R}_u$ and the viscous closure $\boldsymbol{\mathcal{E}}$. The drag force plays a critical role in governing the dynamics of the flow. While relatively accurate empirical models for the mean drag $\langle F_\mathrm{D} \rangle$ are available in the literature [e.g. 228, 268], the deviations from this mean force can be significant. Predicting these deviations given the locations of the surrounding particles is a complex task, which is an area ongoing of research.

Chapter Goals

In this chapter, we assess the capabilities of GP when applied to an unresolved problem related to the Stokes flow of multiple particles. We aim to develop a symbolic model to predict the velocity field around two inline spherical particles, as depicted in Fig. 5.1. An accurate prediction of the velocity field is required to compute the forces acting on a particle in the presence of another

particle, which play a crucial role in the simulation of such flows. Two particles can be considered the simplest case of involving multiple particles, for which no analytical solution is available. We consider this two-particle configuration a test case to evaluate the capabilities and limitations of the proposed GP approach. This serves as a crucial first step toward scaling the method to systems with multiple interacting particles, which is the ultimate objective.

Chapter Overview

First, an overview of state-of-the-art methods from the literature will be given. We then introduce our algorithm designed to tackle the Stokes flow past two spherical particles. Out-of-the-box GP approaches are not sufficient to solve such complex problems. Expert knowledge is required to tailor the algorithm to this specific use-case. To this end, we proposed a novel building block approach, which included the analytical solution for the flow past a single spherical particle as a building block in the function set. Further domain knowledge was integrated on various levels along the pipeline of the algorithm. We employed ground truth data that were generated using the method of Regularized Stokeslets. We performed several experiments with varying distance between the two particles, and compared the results with the reported results of a *multilayer perceptron* (MLP) and a coarse approximation based on the *superimposition method* (SIP). The results demonstrated that our building block approach enables the development of concise solutions. We also observed that MLP-based methods outperformed the equations generated by GP. The evolved models allowed for interpretation, but also revealed limitations, highlighting the need for more robust approaches capable of capturing the complex interactions among particles, especially in scenarios involving more than two particles. These findings provided a strong motivation for the contributions presented in the subsequent chapters.

5.2 Related Research

The following section is largely based on the author's publication [223].

Genetic Programming Approaches

A recent publication by Zille et al. [299] addressed problems at the intersection of GP and fluid mechanics, which are closely related to the topics covered in this thesis. They examined the capabilities of GP algorithms to predict analytical solutions for the Stokes flow around a single spherical particle, which were similar to or derived from Eqs. 5.8 and 5.9. Multiple algorithm variants were assessed, including multiphase GP, unit-aware GP, cooperative coevolution which assembles a solution from multiple evolved subtrees, as well as the inclusion of pre-computed features. The results on six benchmark datasets with known ground truth demonstrated that GP was capable of identifying the correct equations, when a suitable set of optimization objectives was used. The highest accuracy was achieved through the simultaneous optimization of both error and correlation objectives. Introducing additional objectives, such as penalties for unit violations and complexity, did not yield further improvements in terms of accuracy. The importance of incorporating domain knowledge was also evident, as algorithms utilizing expert-designed, precomputed features outperformed the baseline algorithm on more complex datasets. However, a key limitation of the approach was its volatile success rate, with varying results across multiple repetitions of the same algorithm.

A related study by Ross et al. [234] aimed to learn closure models from high-fidelity data on ocean turbulences, which was then employed in low fidelity simulations that required closure. They employed a hybrid GP approach with expert input at predefined stages of the equation development, and were able to identify important building blocks as the main drivers for accurate predictions.

Data-Driven Methods to Predict Deviations from the Mean Drag

In the area of fluid mechanics, more realistic arrangements with a larger number of particles are typically studied. The identification of an accurate model for the hydrodynamic forces in such arrangements is a non-trivial task, as shown

by the rich recent literature on the matter [2, 3, 254, 255]. We are mainly interested in the data-driven approaches, which use highly resolved data to find an accurate model for the deviations from the mean drag $\langle F_\mathrm{D} \rangle$. A promising approach in this area is the *pairwise interaction extended point-particle* (PIEP) method, which assumes pairwise interactions between particles to predict the variation in the drag force [3, 255]. In a flow locally governed by Re and ϕ, the force acting on a particle i can be approximated by aggregating the fluctuations introduced by particles in its neighborhood $\mathcal{N}_\mathrm{i}$. These pairwise interactions depend on the relative positions of the neighboring particles, $\mathbf{r}_j$, where $j \in \mathcal{N}_\mathrm{i}$. In this context, recent literature suggested that the prediction accuracy increased only up to a number of neighboring particles considered between 20 and 30 [3, 255]. Although this assumption introduced some error by accounting only for first-order interactions between particles, their predictive abilities closed the gap in terms of accuracy to those of PR-DNS while maintaining low computational costs. Further notable publications in this area include [185] and [186], which employed multiple linear regression on expansions of spherical harmonics. Seyed-Ahmadi et al. [254] extracted distributions of particle locations within a predefined neighborhood from PR-DNS data. These were used to find correlations between the force exerted on a particle and the locations of its neighboring particles.

ANN-Based Methods to Predict Deviations from the Mean Drag

Balachandar et al. [16] were the first to implement an ANN to predict the variations in the drag. The input features comprised the relative positions of the 15 closest particles within the neighborhood of the particle of interest, as well as Re and ϕ. A single hidden layer with 25 neurons and the hyperbolic tangent as activation function was used. When applied to test data of PR-DNS realizations that had not been included in the training, the ANN exhibited severe overfitting behavior, which the authors believed was due to insufficient training data. Building upon these results, a recent publication by Seyed-Ahmadi et al. [255] indicated that PINNs could overcome the problem of overfitting. A main characteristic of this approach was the parameter sharing between neural network blocks. Following the pairwise interaction assumption, the influence of each neighbor of a particle was calculated by a small ANN, which was shared by all neighbors. The total force on a particle was the linear superposition of the influences of its neighboring particles. In addition to neighboring locations, the predictive features included the local average velocity, which can be approximated empirically from the location of the particles. The PINN approach imposed an underlying form on the model, which was deduced from prior knowledge about pairwise interactions. While this was an important step to align the model with underlying physics, the transformations inside the ANN blocks remained opaque.

Overall, the publication by Zille et al. [299] can be considered valuable preliminary work with respect to the fluid mechanics problems addressed in this thesis. We furthermore observed a gap in the literature between fluid mechanics and GP, with the former studying larger particle arrangements but lacking interpretable models, and the latter offering interpretable models, but lacking studies on the algorithm's performance when applied to complex tasks with unknown ground truth. While Zille et al. [299] applied their algorithm to problems with known ground truth, the focus of this thesis is to develop equations for cases where the ground truth is not known. With the contributions in this chapter, we furthermore aim at making a first step towards interpretable models for more complex arrangements beyond a single particle.

5.3 Proposed Methods

In the following, we first present our algorithm to learn the Stokes flow past two spherical particles. Subsequently, we describe the employed objective functions.

Algorithm 4: Proposed genetic programming algorithm [222]

input : Training Data X, Terminals $\mathcal{T}$, Functions $\mathcal{F}$, phase generations k, crossover probability p_c, mutation probability p_m
output : Set of non-dominated solutions A

$A \leftarrow$ Empty Pareto-dominance based archive
$pop \leftarrow$ Random initial population of solutions
evaluate(pop)
$A \leftarrow$ updateArchive(pop)
repeat
  for k *generations* **do**
    $parents \leftarrow$ select(pop)
    $offspring \leftarrow$ reproduce($parents$, p_c, p_m)
    evaluate($offspring$)
    $pop \leftarrow$ updatePopulation($\cup$(pop, $offspring$))
    $A \leftarrow$ updateArchive($offspring$)
  end
  for k *generations* **do**
    $parents \leftarrow$ select(pop)
    $offspring \leftarrow$ reproduce($parents$, $p_c = 0$, $p_m = 1$)
    evaluate($offspring$)
    $pop \leftarrow$ updatePopulation($\cup$(pop, $offspring$))
    $A \leftarrow$ updateArchive($offspring$)
  end
until *stopping criterion is not reached*
return A

Table 5.1: Problem-dependent sets of terminals and functions for the proposed algorithm [222].

Terminals	Relevant features from training data and set of constants $\{1.0, 2.0, 0.5, 0.25\}$
Functions	$+$, $-$, $\times$, / (protected), $u_0(x, y)$, $v_0(x, y)$, $u_d(x, y)$, $v_d(x, y)$

5.3.1 Overall Algorithm

Multi-Phase Genetic Programming Algorithm

Based on the recent research for predicting the Stokes flow around a single sphere [299], our proposed GP algorithm is outlined in Algorithm 4. Preliminary tests have strongly suggested that a multi-phase GP advanced the evolutionary process in the best case, and did not deteriorate the results in the worst case. Therefore, we enhanced the standard GP procedure by a two phase evolution with k generations of crossover and mutation to generate offspring, followed by k generations of mutation only, similar to the algorithm presented in Sec. 4.3. In this way, we aimed at balancing exploration and exploitation of solutions through genetic operators. This could help to refine solutions that have a suitable structure but require small adjustments on the terminal or function level. We used the $(\mu + \lambda)$ reproduction scheme, which allowed both parents and children to survive to the next generation [222].

Problem-Dependent Terminal and Function Sets

Tab. 5.1 displays the terminals and functions provided to the GP algorithm. As terminals, we selected the relevant features from the training data as well as a few constants. The available functions were basic arithmetic operators, including a protected division to avoid division by zero. In this case, an infinity value was returned. Preliminary experiments indicated that a standard GP setting with only arithmetic operators performed poorly on the given problem, creating the need to incorporate domain knowledge in our approach. As such, we included the solutions for the flow past a single particle as a GP building block, namely the velocities in u- and v-direction: u_0 and v_0 for the disturbance around the left particle, and u_d and v_d for the right particle, with a distance d between them. Given a particle i, the function arguments of u_i and v_i are the position x and y at which the flow is to be predicted. Algorithm 5 outlines this procedure that involves transformation to spherical coordinates and back

Algorithm 5: Computation of flow disturbances u_i, v_i past a single spherical particle at center position $x_{c,i}, y_{c,i}$ [222]

input : x and y coordinates
output: u_i and v_i, $i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20\}$

```
function flowDisturbance(x, y):
    x_rel, y_rel ← compute relative position from particle center x_{c,i}, y_{c,i}
    r, θ ← convert x_rel, y_rel to polar coordinates
    u_r, u_θ ← compute Eqs. 5.8 and 5.9
    u, v ← convert u_r, u_θ to global Cartesian coordinates
    return u − u_∞, v
end
```

to Cartesian coordinates. The computation of the relative position to the center of the particle i is required for the subsequent calculations in spherical coordinates. The functions return the disturbance around a single spherical particle using the Eqs. 5.8 and 5.9. The disturbance describes the variation in the undisturbed flow velocity due to the addition of a particle. It is computed in u direction by $u - u_\infty$, whereas the disturbance in v is not modified, since $||v_\infty|| = 0$ [222].

5.3.2 Objective Functions

Rooted Mean Squared Error

We incorporated the findings of [299] as well as the results presented in Sec. 4.4, and considered multiple objectives in our approach. Our first objective f_1 was the RMSE to minimize the error between the predicted and the observed velocity at specific points in space. It is the primary attribute that determines the quality of a solution [222].

Correlation Coefficient

Furthermore, we employed a rank-based correlation coefficient using the Spearman correlation as a second objective f_2. The intention was to keep individuals in the evolutionary process that were not yet numerically accurate, but produced a high correlation with the target data. Thus, promising individuals had the chance to be refined towards more accurate solutions. To minimize the objective, we defined it as $f_2 = 1 - |\rho|$, where ρ is the Spearman correlation coefficient ranging from -1 to 1. The absolute value was taken to also keep inversely proportional solutions in the population [222].

Dimension Penalty

To account for the compliance with physical laws, which is an essential requirement in equation discovery for science and engineering, we employed a third objective f_3 that penalizes individuals that execute non-physical operations. For each unit-violating operation, a penalty of 1.0 was added to a counter, which was then aggregated throughout an individual. Additionally, to guarantee that the final unit of an individual corresponds to our target unit `meters/second` $= \mathtt{m}^1 \cdot \mathtt{s}^{-1}$, we added the Manhattan distance between the exponents of the SI-base units of which the final unit was composed. For example, the distance of $\mathtt{m}^2 \cdot \mathtt{s}^3$ to the target unit $\mathtt{m}^1 \cdot \mathtt{s}^{-1}$ is 5. The constants in the terminal set were considered as dimensionless quantities [222].

5.4 Experiment Setup

In this section, the two particles problem from the area of fluid mechanics is defined, as well as the baseline methods to which the proposed GP algorithm is compared. Furthermore, we will describe the algorithm settings and criteria to assess the quality of the solutions developed by GP.

5.4.1 Two Particles Benchmark

General Benchmark Setup

The goal of this chapter is to make a first step towards closing the litera-

ture gap between GP and fluid mechanics, and assess the performance of the GP algorithm on the Stokes flow past two spherical particles, for which no ground truth equations were known. To identify the potentials and limitations of GP on the two-particle problem, we introduced a new benchmark dataset with varying distances between the particles (see Fig. 5.1). As the particles move closer together, their mutual influence increases, and vice versa. Building on this concept, the larger the distance between the two particles, the more accurately the surrounding flow can be approximated by superimposing the analytical solution for a single particle. We anticipated GP to perform well on a large distance $d >= 10$ and wanted to identify limitations of the approach by a stepwise reduction of the distance. To this end, training data for $d = [20, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$ was generated. Since the flow velocity is described by two components, u and v, separate benchmarks were created. The combinations of eleven distances and two velocity components made a total of 22 benchmarks. Each dataset contained the following features [222]:

- u, v [$\mathrm{m\,s^{-1}}$]: target velocities in u or v direction
- x, y [m]: positions in for which the flow is to be predicted
- x_{c1}, y_{c1}, x_{c2}, y_{c2} [m]: center coordinates of the two spheres
- d [m]: distance between the two spheres
- a [m]: radius of the spheres
- u_∞ [$\mathrm{m\,s^{-1}}$]: undisturbed flow velocity

Data Generation with Regularized Stokeslets

The datasets were generated using the regularized Stokeslets as introduced in Sec. 5.2, for which the full implementation details can be found in the author's publication [222]. We distributed $N = 5000$ Stokeslets over the surface of a sphere following the generalized spiral set approach. The parameters were set to a particle radius of $a = 0.25$ and the undisturbed velocity $||u_\infty|| = 1.0$. The origin of the coordinate frame was located at the center of the left sphere. The right sphere had the center coordinates $(d, 0)$ so that the particles were aligned along the direction of the free stream, as depicted in Fig. 5.1. To cover all relevant flow disturbances, the training data comprised $[x_{c,0} - 4a, x_{c,d} + 4a]$ in x-direction and $[y_{c,0} - 4a, y_{c,0} + 4a]$ in y-direction. We employed a resolution of 200 data points in x- and 50 data points in y-direction, which made a total of 10,000 instances subtracting the data points that lay inside the spheres, for which no flow could be predicted [222].

5.4.2 Baseline Methods

Fluid Mechanics Baseline: Superimposition Method

To compare the accuracy of our GP algorithm to methods from the literature, we also assessed the performance of two baseline methods on the introduced datasets. First, we employed the *superimposition method* (SIP), which can be considered a coarse estimator from the area of fluid mechanics. It approximates the velocity field at any point by summing the individual disturbances caused by each particle in the domain at that point. Similarly to our approach, it uses the analytical solution of the flow around a particle to calculate the disturbances caused by M particles [222]:

$$u(x, y) = u_\infty + \sum_{i=1}^{M} u_{\text{disturbance},i}(x, y) \tag{5.11}$$

$$v(x, y) = \sum_{i=1}^{M} v_{\text{disturbance},i}(x, y) \tag{5.12}$$

Since the disturbances in u and v direction were also contained in the function set of our proposed GP, we could directly compare the two methods. Due to

Table 5.2: GP algorithm parameters [222].

Parameters	Settings
μ	2,000
λ	2,000
Number of Evaluations	600,000
Generations per Phase k	20
Selection Mechanism	NSGA-II selection
Initialization Method	Half Full, Half Grow
Crossover Probability p_c	0.7
Mutation Probability p_m	0.3
Crossover	One-point, Leaf-biased ($p_{leaf} = 0.9$) (chosen at random)
Mutation	Uniform, Insert, Shrink, Node Replacement (chosen at random)
Mutation ($p_m = 1.0$)	Shrink (1/3), Node Replacement (2/3)
Max. Tree Length	30
Max. Init Depth	4
Min. Init Depth	1
Max. Mutation Depth	2
Min. Mutation Depth	0
Objectives	f_1, f_2, f_3
Train-Test Ratio	0.7 train, 0.3 test
Runs	31

the deterministic nature of the superimposition method, we executed it only once.

ML Baseline: Multilayer Perceptron

Additionally, we incorporated an MLP as the baseline method from the ML field. To this end, we used the `Scikit-Learn` implementation of the MLP regressor. We employed two fully connected hidden layers with 20 neurons each and trained the network for 200 epochs. The input features were the relative positions of the centers of the two particles to the location of interest within the velocity field, namely $x_{1,\mathrm{rel}}$, $y_{1,\mathrm{rel}}$, $x_{2,\mathrm{rel}}$ and $y_{2,\mathrm{rel}}$, as well as the particle radius a, and distance d between the particles. Preliminary experiments indicated that also small network configurations were capable of approximating the velocity fields around two spherical particles accurately. Consequently, only one MLP was trained on all datasets combined, and the distance between the particles was included as an additional input feature.

5.4.3 Algorithm Settings

Various algorithm parameters had to be defined for GP to yield satisfactory results (see Tab. 5.2). We employed a similar parameter setting as in the preliminary work for the Stokes flow around one particle [299]. To limit individual lengths and avoid bloating, a length limit (i.e., the number of nodes in the GP tree) of 30 was applied. The datasets were split with a ratio of 70% training data and 30% test data. For each benchmark instance, 31 repetitions of the GP algorithm were performed [222]. The GP algorithms were implemented using the `DEAP`-framework version 1.3.1 [90] and the `pint` package version 0.16.1.

5.4.4 Quality Assessment

Selection Criteria for Final Solutions

During the evolutionary process, the algorithm optimized three objectives, namely the RMSE f_1, correlation f_2 and dimension penalty f_3. Additionally, we recorded the *coefficient of determination* (R^2) as well as the *mean absolute error* (MAE) during the training process. Since the algorithm optimized for multiple objectives, a single optimal solution could not be identified anymore. Thus, we used the following procedure to determine the final solution of a run:

Table 5.3: Overview of domain knowledge integrated as bias into the GP algorithms proposed in this chapter.

Bias Type	Bias
Inductive	Problem-specific set of building blocks included in the function set (see Sec. 5.3.1)
Learning	Conversion between spherical and Cartesian coordinate systems to simplify the problem for the algorithm (see Algorithm 5)
Learning	Dimension penalty to punish equations with unit violations (see Sec. 5.3.2)

To guarantee the explainability of the solutions, we only considered solutions that complied with physical laws in our final evaluation, i.e., that had a dimension penalty $f_3 = 0$. From those solutions, the one with the lowest f_1 was the designated output of the run [222].

Statistical Comparison

To verify if the proposed and the baseline methods achieved results that were significantly different, a Friedman's test was performed. If the equality hypothesis was rejected, the Holm-Bonferroni test for adapted p-values was conducted for pairwise comparison of the methods. A significance level $\alpha = 0.05$ was employed. Tab. 5.3 summarizes the types of domain knowledge included as bias in the proposed algorithms and experiment settings.

5.5 Results and Analysis

5.5.1 Convergence Behavior

The convergence behavior of GP for selected distances in u direction is displayed in Fig. 5.2. The behavior in v direction is similar to the ones shown here. Each line represents the mean RMSE over the best solutions for 31 runs at different times, where "best" refers to the solution with the lowest RMSE and a dimension penalty of $f_3 = 0$. The performance on the training set is depicted with a solid line and on the test set with a dashed line.

No overfitting could be observed for GP and the two baseline methods.

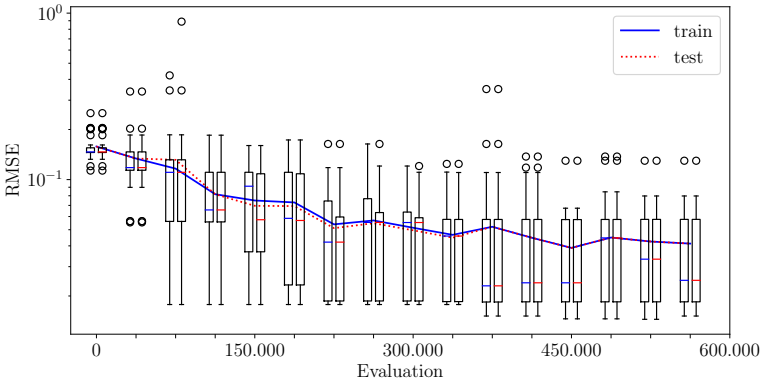
The three plots indicate that no overfitting to the training data occurred, as training and test error decreased simultaneously and no increase in the test error in later stages of the training could be identified. On the contrary, the gap between training and test error is almost negligible. The convergence speed for $d = 10$ is notably higher than for $d = 1$ and $d = 5$, implying that accurate solutions were easily found. The curve for $d = 10$ exhibits a convergence behavior that is nearly optimal, with a steep decline in the first quarter of the training time and a flat curve in the remainder. The learning curves for $d = 1$ and $d = 5$ do not show a perfect convergence, and it could be anticipated that further reduction RMSE could be further reduced with more evaluations. Furthermore, two spikes can be identified in the error curve for $d = 5$. This can primarily be explained with the outliers of one out of 31 runs run, which had a considerably larger error value compared to the others [222].

The RMSE distributions exhibited high variability over multiple runs.

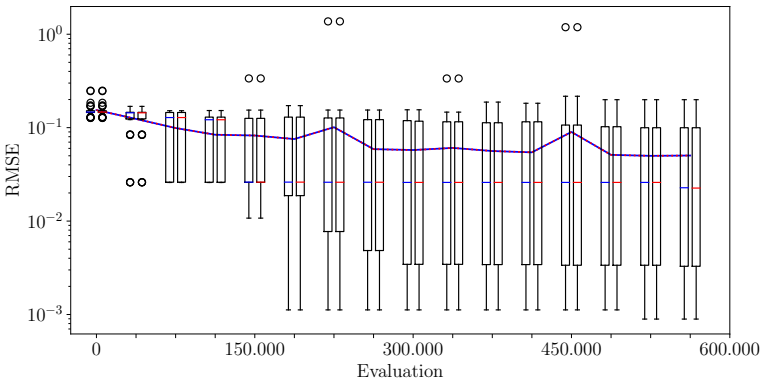
Throughout all of the 31 runs, it can be observed that the RMSE values were distributed over a relatively wide range, even at the end of the training phase. We can interpret this in two ways: First, the results might be improved further if the training duration is increased; and second, it is possible that the algorithm frequently revisited the same local optima and struggled to leave them towards better solutions. Since some runs identified solutions with low RMSE values, we know that these solutions existed tended to be difficult to obtain when the algorithm was executed multiple times. A closer look at the population dynamics and the specific equations in the population at different generations is required to better understand this observation.

Figure 5.2: Convergence behavior of GP on training and test data for selected distances d in u direction on a logarithmic y-axis. The lines display the mean RMSE, and boxplots the distribution over 31 runs.

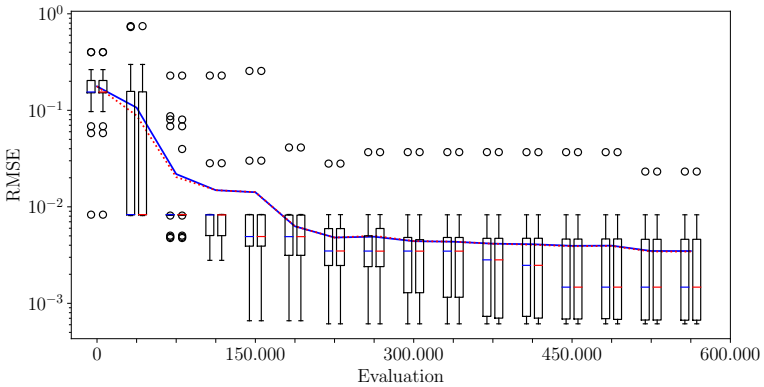
(a) Distance between particles $d = 1$.



(b) Distance between particles $d = 5$.



(c) Distance between particles $d = 10$.



5.5.2 Effects of Different Distances Between the Particles

Unit compliant equations were found by GP.

One goal of this chapter is to assess the limitations of the proposed GP algorithm and compare the performance in terms of decreasing distances between the two particles. We hypothesize that the algorithm will demonstrate better performance at larger distances between particles, where the mutual interactions are weaker. For all runs of each benchmark instance, at least one individual with a dimension penalty of $f_3 = 0$ could be obtained. Thus, unit compliant solutions could be identified in general for the given problem. The performance of the proposed algorithm, along with the baseline methods, on the test dataset is summarized in Tabs. 5.4 and 5.5, reporting both R^2 and RMSE metrics. To maintain clarity and conciseness, we have omitted the MAE results, as RMSE provides a more rigorous measure of solution quality, which penalizes larger errors more. The values of objectives f_2 and f_3 are also omitted in the table, as they are primarily used as supporting objectives to enhance the training

Table 5.4: Results of experiments for the u component of the flow [222].

		R^2			RMSE		
d	Method	Best	Mean ± Std	SC	Best	Mean ± Std	SC
1	GP	0.98898	0.93829 ± 0.09497		0.01550	0.04124 ± 0.02642	
	SIP	0.64865	0.64865 ± 0.00000	−	0.18567	0.18567 ± 0.00000	−
2	GP	0.99762	0.89169 ± 0.16221		0.00483	0.03873 ± 0.03515	
	SIP	0.81471	0.81471 ± 0.00000	−	0.09112	0.09112 ± 0.00000	−
3	GP	0.99099	0.76842 ± 0.26984		0.00264	0.05576 ± 0.04611	
	SIP	0.90174	0.90174 ± 0.00000	+	0.05440	0.05440 ± 0.00000	=
4	GP	0.99343	0.64824 ± 0.35060		0.00114	0.06542 ± 0.05365	
	SIP	0.94880	0.94880 ± 0.00000	+	0.03637	0.03637 ± 0.00000	+
5	GP	0.99586	0.72624 ± 0.38381		0.00097	0.05048 ± 0.05600	
	SIP	0.97220	0.97220 ± 0.00000	+	0.02606	0.02606 ± 0.00000	+
6	GP	0.99886	0.92884 ± 0.16668		0.00072	0.01683 ± 0.03354	
	SIP	0.98411	0.98411 ± 0.00000	=	0.01964	0.01964 ± 0.00000	=
7	GP	0.99918	0.97034 ± 0.11585		0.00061	0.01045 ± 0.02073	
	SIP	0.99036	0.99036 ± 0.00000	=	0.01531	0.01531 ± 0.00000	=
8	GP	0.99938	0.94054 ± 0.19538		0.00059	0.01269 ± 0.03218	
	SIP	0.99381	0.99381 ± 0.00000	=	0.01225	0.01225 ± 0.00000	=
9	GP	0.99749	0.99165 ± 0.00682		0.00059	0.00371 ± 0.00441	
	SIP	0.99586	0.99586 ± 0.00000	+	0.00998	0.00998 ± 0.00000	−
10	GP	0.99878	0.99253 ± 0.00686		0.00061	0.00340 ± 0.00415	
	SIP	0.99714	0.99714 ± 0.00000	+	0.00828	0.00828 ± 0.00000	−
20	GP	0.99927	0.99865 ± 0.00026		0.00029	0.00085 ± 0.00051	
	SIP	0.99982	0.99982 ± 0.00000	+	0.00188	0.00188 ± 0.00000	−
all	MLP	0.99811	0.99545 ± 0.00202	+	0.00010	0.00022 ± 0.00010	+

Table 5.5: Results of experiments for the v component of the flow [222].

		R^2			RMSE		
d	Method	Best	Mean ± Std	SC	Best	Mean ± Std	SC
1	GP	0.99358	0.92330 ± 0.06684		0.00483	0.01703 ± 0.00921	
	SIP	0.92054	0.92054 ± 0.00000	=	0.02633	0.02633 ± 0.00000	−
2	GP	0.99604	0.99324 ± 0.01196		0.00201	0.00297 ± 0.00272	
	SIP	0.97546	0.97546 ± 0.00000	−	0.01161	0.01161 ± 0.00000	−
3	GP	0.99704	0.99370 ± 0.00451		0.00101	0.00311 ± 0.00263	
	SIP	0.98821	0.98821 ± 0.00000	−	0.00743	0.00743 ± 0.00000	−
4	GP	0.99735	0.99525 ± 0.00208		0.00072	0.00257 ± 0.00183	
	SIP	0.99317	0.99317 ± 0.00000	−	0.00531	0.00531 ± 0.00000	−
5	GP	0.99752	0.99668 ± 0.00109		0.00058	0.00160 ± 0.00132	
	SIP	0.99559	0.99559 ± 0.00000	−	0.00401	0.00401 ± 0.00000	−
6	GP	0.99843	0.99705 ± 0.00065		0.00049	0.00155 ± 0.00090	
	SIP	0.99695	0.99695 ± 0.00000	=	0.00316	0.00316 ± 0.00000	−
7	GP	0.99760	0.99711 ± 0.00033		0.00051	0.00133 ± 0.00081	
	SIP	0.99777	0.99777 ± 0.00000	+	0.00256	0.00256 ± 0.00000	−
8	GP	0.99777	0.99736 ± 0.00022		0.00046	0.00128 ± 0.00060	
	SIP	0.99831	0.99831 ± 0.00000	+	0.00212	0.00212 ± 0.00000	−
9	GP	0.99829	0.99744 ± 0.00024		0.00045	0.00113 ± 0.00056	
	SIP	0.99869	0.99869 ± 0.00000	+	0.00179	0.00179 ± 0.00000	−
10	GP	0.99822	0.99747 ± 0.00016		0.00036	0.00110 ± 0.00043	
	SIP	0.99894	0.99894 ± 0.00000	+	0.00154	0.00154 ± 0.00000	−
20	GP	0.99922	0.99921 ± 0.00001		0.00044	0.00054 ± 0.00005	
	SIP	0.99974	0.99974 ± 0.00000	+	0.00057	0.00057 ± 0.00000	−
all	MLP	0.99440	0.98840 ± 0.00434	+	0.00002	0.00004 ± 0.00001	+

process and facilitate the interpretation of the solutions. For each measure, the value of the best solution, the mean over the 31 runs, and the standard deviation to determine the stability of an algorithm are shown. The baseline methods were subjected to a statistical comparison (SC) with GP as described in Sec. 5.4.4. The symbols (=), (-), and (+) are used to indicate whether there were no significant differences with GP, whether the results were significantly worse or whether they were significantly better, respectively.

MLP always outperformed GP and SIP.

The columns *Best* in Tabs. 5.4 and 5.5 indicate that GP found at least one solution for all benchmark instances that was better than the fluid mechanics baseline SIP. This observation is especially interesting for fluid dynamics experts, for whom a single optimal solution over multiple runs is already beneficial [222]. Furthermore, it became apparent that the simple MLP always performed by orders of magnitude better than both the SIP and the overall best solution obtained by GP over all benchmark datasets.

GP achieved stable results in u direction for distances $d \geq 6$.

Fig. 5.3 visually supports the results from these tables and displays the RMSE distributions over the 31 runs for GP and MLP as well as the deterministic result of SIP. As expected, the magnitude of the error decreased with increasing

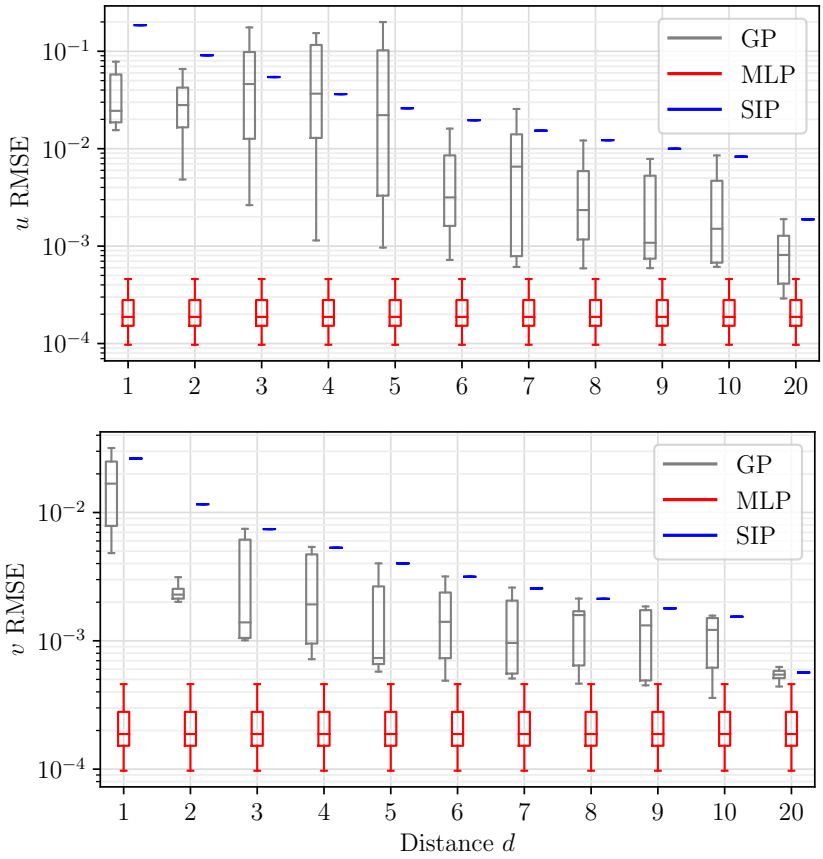


Figure 5.3: RMSE distribution for GP, MLP and SIP in u and v direction over 31 independent runs on a logarithmic y-axis.

distances between the two particles. In the u direction from $d = 20$ to $d = 6$, GP produced solutions that were better than, or at least as good as, those obtained by SIP. For $d < 6$, the error spread increased, and stable results could not be guaranteed, which is also reflected in the statistical comparison of RMSE values. Similarly, the mean values of R^2 for GP were greater than 0.9 for $d >= 6$, with a decrease for $d = [5, 4, 3]$. Interestingly, GP found comparatively good solutions for small distances $d = 1$ and $d = 2$ compared to SIP. This can be explained by the fact that SIP is a deterministic approach and the same equation was used on all benchmark sets. GP, on the other hand, was trained separately on each benchmark set and could therefore adapt to the changing flow pattern for different distances between the particles [222].

GP outperformed SIP in v direction.

In the direction v, the GP algorithm outperformed SIP for all distances in terms of RMSE. Although the statistical comparison of R^2 indicated statistically better results in favor of SIP for some benchmark instances, the real values of R^2 of 0.99 do not reflect large differences between the two methods. The better performance in the v direction can be explained by the fact that the solution required fewer operations than in the u direction, i.e., u_∞ was omitted. Again, the MLP surpassed GP in all benchmark problems.

5.5.3 Explainability of Solutions

Selection and Simplification Procedure

A main concern of these experiments was the explainability of the solutions produced by the proposed GP algorithm. Objective f_3 guaranteed the conformity with physical laws. Tabs. 5.6 and 5.7 list the best GP solutions over 31 runs for the u and v components of the flow for different distances. We used the Python `sympy` package to simplify the equations.

Building blocks were used in final equations.

All equations utilized the solutions for the disturbance around a single particle, i.e., the expert knowledge given to the algorithm. All individuals followed a similar scheme of adding one or multiple terms to the undisturbed flow velocity u_∞. Essentially, the disturbances around the two particles were aggregated.

Table 5.6: Best solutions in u direction [222].

d	Solution
1	$u_\infty + 2.0\,\mathrm{u}_0\left(\frac{4.0au_\infty}{\mathrm{u}_0\,(x,y)+\mathrm{u}_1\,(x,y)}, 0.25y\right)$
2	$u_\infty + \mathrm{u}_0\,(x,y) - \mathrm{u}_0\left(y, \frac{2.5au_\infty}{0.5\,\mathrm{u}_0\,(x,y)+0.5\,\mathrm{u}_2\,(x,y)}\right) + \mathrm{u}_2\,(x,y)$
3	$u_\infty + 0.9\,\mathrm{u}_0\,(x,y) + 0.9\,\mathrm{u}_3\,(x,y)$
4	$u_\infty + 0.923\,\mathrm{u}_0\,(x,y) + 0.923\,\mathrm{u}_4\,(x,y)$
5	$u_\infty + \frac{u_\infty(\mathrm{u}_0\,(x,y)+\mathrm{u}_5\,(x,y))}{u_\infty-\mathrm{u}_5\,(-2.75a,y)}$
6	$u_\infty + \frac{(u_\infty+\mathrm{u}_6\,(-6.0a,y))(\mathrm{u}_0\,(x,y)+\mathrm{u}_6\,(x,y))}{u_\infty}$
7	$u_\infty + 0.958\,\mathrm{u}_0\,(x,y) + 0.958\,\mathrm{u}_7\,(x,y)$
8	$u_\infty + \frac{\mathrm{u}_0\,(x,y)+\mathrm{u}_8\,(x,y)}{1.0-\frac{0.75\,\mathrm{u}_8\,(a,0.25y)}{u_\infty}}$
9	$u_\infty + 0.970\,\mathrm{u}_0\,(x,y) + 0.970\,\mathrm{u}_9\,(x,y)$
10	$u_\infty + \mathrm{u}_{10}\,(x,y) + \frac{(u_\infty+0.75\,\mathrm{u}_{10}\,(x,y))\,\mathrm{u}_0\,(x,y)}{u_\infty}$
20	$u_\infty + \mathrm{u}_0\,(x,y) + \mathrm{u}_{20}\,(x,y) + \frac{\mathrm{u}_0\,(2.25x,2.0y)\,\mathrm{u}_{20}\,(x,y)}{u_\infty}$

Table 5.7: Best solutions in v direction.

d	Solution
1	$-0.25\,\mathrm{v}_0\,(x,y) + \mathrm{v}_0\,(0.125a+x, 1.0y) + \mathrm{v}_1\,(x,y) - 0.25\,\mathrm{v}_1\,(0.25a+x,y)$
2	$0.842\,\mathrm{v}_0\,(y,x) + 0.842\,\mathrm{v}_2\,(x,y)$
3	$\frac{u_\infty(\mathrm{v}_0\,(x,y)+\mathrm{v}_3\,(x,y))}{u_\infty-\mathrm{v}_0\,(a,1.75a)+\mathrm{v}_3\,(2x,0)}$
4	$\frac{u_\infty(\mathrm{v}_0\,(x,y)+\mathrm{v}_4\,(x,y))}{u_\infty-\mathrm{v}_0\,(2.5a,3.0a)}$
5	$\mathrm{v}_0\,(x,y) - 0.0625\,\mathrm{v}_0\,(y,x) + 0.9375\,\mathrm{v}_5\,(x,y)$
6	$0.947\,\mathrm{v}_0\,(x,y) + 0.947\,\mathrm{v}_6\,(x,y)$
7	$-0.045\,\mathrm{v}_0\,(x,1.0y) + \mathrm{v}_0\,(y,x) + 0.954\,\mathrm{v}_7\,(x,y)$
8	$0.961\,\mathrm{v}_0\,(x,y) + 0.961\,\mathrm{v}_8\,(x,y)$
9	$0.968\,\mathrm{v}_0\,(x,y) + 0.968\,\mathrm{v}_9\,(x,y)$
10	$0.969\,\mathrm{v}_0\,(y,x) + 0.969\,\mathrm{v}_{10}\,(x,y)$
20	$\frac{\mathrm{v}_0\,(x,y)+\mathrm{v}_{20}\,(x,y)}{1.0+\frac{3.0\,\mathrm{v}_{20}\,(y,-a+x)}{u_\infty}}$

This pattern is very similar to the superimposition method, which adds the disturbances of all particles involved. However, GP found slight modifications to the superimposition method to generate better results, such as multiplying the solution around a single particle $\mathrm{u_d}$ for $d = 9$ with a factor of 0.97, $d = 7$ (factor 0.958), $d = 4$ (factor 0.923), and $d = 3$ (factor 0.9). Interestingly, instead of using such a fixed factor, other solutions included a term that depends on x and y, which produced infinitesimal values, such as $\frac{\mathrm{u}_0\,(2.25x,2.0y)\,\mathrm{u}_{20}\,(x,y)}{u_\infty}$ in the solution for $d = 20$. Overall, the solutions followed a pattern similar to the superimposition method with slight modifications. The terms involved were physically meaningful and explainable, while the solutions are concise throughout all benchmark instances [222].

5.6 Discussion and Limitations

Limitations of GP Compared to Baseline Methods

The overall results indicate that the tested building block approach for GP was outperformed by a small MLP with two hidden layers and 20 neurons each, both in the u and v directions of the flow and for all distances between the two particles. The GP algorithm identified at least one solution that achieves lower RMSE values compared to SIP, while delivering concise, unit conformal equations. Compared to the baseline methods, the models evolved with GP exhibited a higher variability in terms of RMSE over multiple realizations. The generated equations were structured similarly to a linear regression model, where the algorithm fits coefficients to the provided building blocks. However,

unlike conventional regression, the constants used in the equations were assembled from the predefined set of constants in the terminal set, which limits their range of potential values. In the future, one could consider a regression algorithm on top of GP to allow for better fitting of constants.

Limitations with Increasing Problem Complexity

The limitations of our algorithm were primarily determined by the velocity component with the weaker prediction accuracy, as accurate flow prediction requires both components to be reliably estimated. Overall, the results in the u direction suggest that the proposed GP algorithm provides stable outcomes with low failure rates for datasets where the distance between particles is $d \geq 6$. While these results are a first step towards identifying the velocity field in Stokes flow involving more than one particle, we could identify a few issues that limited the applicability of this method to more complex arrangements. One key challenge was that the accuracy of the evolved equations decreased as the distance between particles decreased. In other words, the higher their mutual influence, the worse the algorithm performed at predicting the disturbance in the velocity field introduced by the particles and their interactions. It can be assumed that an increased number of particles would further increase the difficulty of the problem. Therefore, the current approach may not be suitable for accurately predicting velocity fields involving more than two particles.

From Prediction of Velocity Fields to Direct Force Prediction

Ultimately, the prediction of a velocity field is an intermediate step to computing the hydrodynamic forces acting on a particle, which are relevant for accurate simulations. Velocity fields are complex to predict, resembling a continuous distribution of velocities throughout the flow domain, meaning that the prediction involves estimating values across multiple spatial points in a potentially large region. Thus, shifting the focus to predicting the forces directly might offer a more manageable approach, estimating an aggregate value that encapsulates the overall effect of the flow on each particle. Given the outstanding prediction accuracy of MLPs demonstrated in this chapter, and the potential to model particle-laden flows as a graph of interacting entities, it seems promising to combine the learning power of MLPs with the expressiveness of GP and to apply GNNs to solve the problem.

5.7 Summary

This chapter presented a first study on the prediction of the Stokes flow around two inline spherical particles using GP for symbolic regression. We implemented a multi-objective approach originally designed for the flow around one particle. Preliminary trials showed that GP has difficulties finding a regression function with standard GP parameter settings. To enrich the algorithm with expert knowledge, the solution for the flow around one particle was given to the function set. To identify the strengths and limitations of the GP algorithm, we proposed a new benchmark for the u and v velocity components of the flow, with decreasing distances between the particles. Our multi-objective approach allowed for solutions aligned with physical laws, which contributes positively to the explainability of the final solution set. All the solutions were concise and physically meaningful. The algorithm successfully included the analytical solution for one particle in the prediction for the flow around two particles. GP found at least one solution that outperformed the SIP baseline method, and achieved significantly better results for distances $d >= 6$. An MLP outperformed the GP algorithm throughout all benchmark instances. Many of the GP solutions multiplied the solution for one particle with a constant and aggregated them. Currently, these constants are built from combinations of values in the terminal set. We observed limitations of the algorithm in terms of increasing problem complexities, in terms of repetition stability and the constants used. The remainder of this thesis will address these issues, which largely overlap with the limitations identified in the previous Chapter 4.7.

Part III

Algorithmic Advances

6 Multi-Objective Island Model Genetic Programming

Introduction and Chapter Goals

As demonstrated in the preceding two chapters, the RMSE values of the GP algorithms exhibited considerable variability when they were executed multiple times. This behavior was observed in both the inverse kinematics and fluid mechanics benchmarks. As pointed out in Sec. 3.2.3, distributed models or *island model*s (IMs) is a widely used method for overcoming these issues and achieving higher success rates. At the same time, we identified a gap in the literature regarding the interplay between multi-objective GP, as it is predominantly implemented in practical applications like ours, and the parameter settings and performance of IMGP. While the literature typically studies single-objective problems, this chapter assesses the impact of combinations of different objective functions and IM configurations on enhancing the success probability of an algorithm. Our goal is to deduce optimal algorithmic configurations, which can be utilized in the future to discover unknown equations.

This chapter is largely based on the author's publication [225].

6.1 Proposed Methods

In the following, we introduce algorithmic components that were examined in our experiments. We assessed their performance on known benchmark equations to identify algorithm configurations with high success rates.

6.1.1 Island Model Configurations

Specification of Migration Topologies

As pointed out in Sec. 3.2.3, IM algorithms have numerous hyperparameters which potentially affect their performance. An extensive study in a grid-search-fashion over multiple values for all hyperparameters exceeds the scope of this thesis. Therefore, we adopted some hyperparameters that performed well in related research, and implemented them in a multi-objective algorithm. Motivated by [57], we investigated the performance of archive-based migration compared to non-archive based migration. To this end, we considered two migration topologies:

- Archive-based migration (A): The best k individuals of all subpopulations build an archive of overall best solutions. For each subpopulation of size m, we replace k individuals at random with k individuals from the archive. Fig. 6.1a displays the archive-based migration topology.

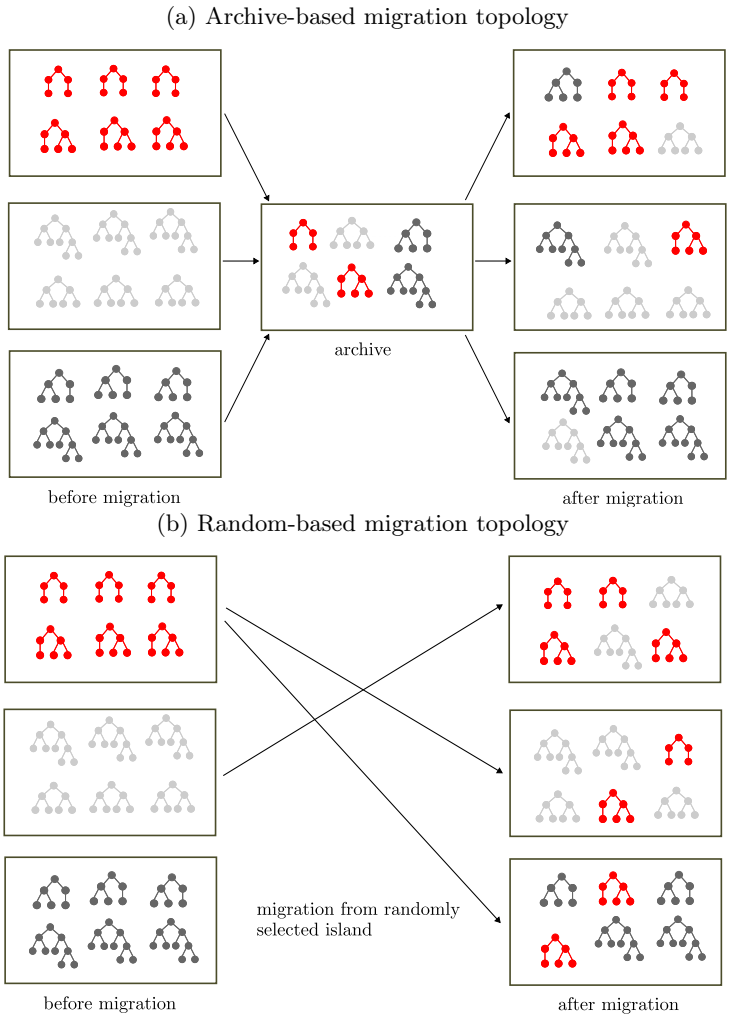


Figure 6.1: Overview of assessed migration topologies [225].

- Random migration (R): In the random migration topology, the subpopulation from which individuals migrate to another is chosen at random. The k worst individuals of the respective island are replaced with the k best individual of the randomly selected island. Due to the random nature of the migration, multiple subpopulations can receive individuals from the same island, as depicted in Fig. 6.1b.

Specification of Migration Rates

Each migration can increase the diversity of solutions within a subpopulation. Usually, a fixed migration rate is used in an IM algorithm, so that migrations are uniformly distributed over the algorithm runtime. In preliminary trials, we observed that some algorithms performed better when migrations were only performed in the first two thirds of the algorithm runtime. The final third of the experiment, which occurred without migration, may have enabled the algorithm to reach a state of convergence without the introduction of further disturbances from migration. Therefore, in addition to the migration topology, we also investigated the distribution of migrations over the evolution [225].

The Overall Algorithm

Algorithm 6 outlines the proposed IM algorithm. The migration schedule m_{sched} contains the generation indices in which migrations are performed. The migration function in line 12 either calls the archive-based or random migration topology. The algorithm returns a *Hall of Fame* (HOF), which contains the set of all non-dominated solutions that have been obtained throughout the runtime of the algorithm.

6.1.2 Objective Functions

Error Objective

Inspired by the approach of Zille et al. [299], we adopted a multi-objective

Algorithm 6: Proposed island model algorithm [225].

input : training data X, number of subpopulations m, subpopulation size p, number of generations n, migration schedule m_{sched}
output : hall of fame H

```
pops ← set of m random initial subpopulations, each of size p
evaluate(pops)
H ← updateHOF(pops)
for gen = 1...n do
    for i = 1...m do
        parents ← select(pops[i])
        offspring ← reproduce(parents)
        evaluate(offspring, X)
        pops[i] ← updatePopulation(∪(pops[i], offspring))
    end
    if gen ∈ m_sched then
        pops ← migrate(pops)
    end
    H ← updateHOF(pops)
end
return H
```

perspective in our methodology. Our primary objective, denoted by f_1, minimizes the error between the prediction and the target variable. To quantify this error, we employed the RMSE, a widely used fitness measure for GP, which emphasizes larger errors with greater penalties.

We want to point out that the error objective used in [299] was, contrary to ours, the *maximum absolute error* (MaxAE). Zille et al.'s preliminary experiments demonstrated that MaxAE yielded superior performance on the given benchmark equations compared to RMSE. However, the MaxAE should be used with caution when unknown equations are discovered from data. Experimental data often contains noise and outliers, and MaxAE tends to put higher emphasis on minimizing the error on the outliers, rather than finding an equation that performs generally well across the entire dataset. Since we aimed at deducing a robust algorithm configuration to discover unknown equations in the future, we employed the RMSE in our experiments [225].

Correlation Objective

We adopted the correlation objective as well as the dimension penalty as presented in Chapters 4.3.1 and 5.3.2, thus, these are only briefly introduced here. We incorporated the rank-based Spearman correlation as our second objective f_2 in the evolutionary process. The purpose was to enable individuals with a strong correlation with the target variable to survive to the next generation, even when they performed poorly on the error objective f_1. This approach enables promising individuals to undergo refinement, and guides the search towards more accurate solutions. The formulation $f_2 = 1 - |\rho|$ was used as a minimization objective, where ρ represents the Spearman correlation coefficient, with a potential output range from -1 to 1 [225].

Dimension Penalty Objective

The third objective was a dimension penalty, which penalizes individuals with non-physical operations. When a unit-violating operation occurred, the penalty value n_{viol} was increased by 1.0 for all operations within an individual. Moreover, this objective had the purpose of aligning the output unit of a solution with the target unit, which was in our case $\texttt{meters/second} = \texttt{m}^1 \cdot \texttt{s}^{-1}$. The Manhattan distance between the exponents of the SI-units of the predicted and target units was added to the penalty value, resulting in $f_3 = n_{\text{viol}} + ||\hat{u}_{SI} - u_{SI}||_1$. Both, f_2 and f_3, were supporting objectives, i.e., we assessed whether the inclusion of these objectives improved the performance with regard to f_1 [225].

Table 6.1: Benchmark equations [225].

Equation	Training Features	Target Variable	# Samples
6.1	u_r, u_θ, θ	u_x	366
6.2	u_∞, a, r, θ	u_r	366

6.2 Experiment Setup

6.2.1 Benchmark Datasets

Datasets for the Stokes Flow Around a Sphere

We conducted experiments to assess the effects of various combinations of objective functions and IM configurations on the success rate of a GP algorithm. To this end, we employed the following fluid mechanics benchmark functions describing velocity fields corresponding to the Stokes flow around a single spherical particle:

$$u_x = u_r \cdot \cos(\theta) - u_\theta \cdot \sin(\theta) \tag{6.1}$$

$$u_r(r,\theta) = u_\infty \cos\theta \left(1 + \frac{a^3}{2r^3} - \frac{3a^2}{r}\right) \tag{6.2}$$

We selected these equations because they have previously shown high variations in the results between multiple runs [299]. Tab. 6.1 summarizes the characteristics of the benchmark datasets. Each dataset was split with a ratio of 80% for training and 20% for testing [225].

6.2.2 Algorithm Variants

Specification of IM Parameters

Furthermore, we assessed the impact of different IM configurations. As mentioned earlier, we could not capture the entire parameter space of IM algorithms in this chapter. We compared the performance of a single-population algorithm, denoted by S, to IM algorithms with $m \in [5, 10]$ subpopulations. Similar configurations were used in related works [83, 198]. The migrations were executed according to the random (R) and archive-based (A) migration topologies as defined in Sec. 6.1.1. The number of migrations was set to ten, which were distributed over the algorithm runtime according to two schemes. In the first variant, they were distributed uniformly, denoted by D0. In the second variant, they only happened in the first two thirds of the algorithm runtime, denoted by D1. To facilitate comparison between the single-population and IM algorithms, the population size of 2000 individuals was evenly distributed among the respective number of islands. A subpopulation thus constituted $\frac{2000}{m}$ individuals [225]. The proportion of migrating individuals was set to 0.035, which was also used in a related study [57].

Specification of General Algorithm Parameters

We compared the performance of one single-objective and three multi-objective functions as specified in Sec. 6.1.2 with each other: f_1, $f_1 f_2$, $f_1 f_3$, $f_1 f_2 f_3$. All experiments employed the function set $\mathcal{F} = \{+, -, \cdot, \circ^2, \circ^3, \frac{1}{\circ}, \sin(\circ), \cos(\circ), \sqrt{\circ}\}$ and the terminal set $\mathcal{T} = \mathcal{C} \cup \{\text{training features}\}$, where $\mathcal{C} = \{4, 3, 2, 1, \frac{1}{2}, \frac{1}{4}\}$. The experiments were performed for 800 generations. In summary, we tested four objective functions with one single-population and eight IM configurations, which made a total of 36 algorithm variants. To compare the effects of the assessed algorithmic variants, we used the same random seed in each of the 31 algorithm executions across all algorithm configurations. Tab. 6.2 summarizes additional settings of the GP algorithms used in our experiments, which were implemented in the Python library `DEAP` [90].

Table 6.2: GP algorithm parameters [225].

Parameter	Settings
Population Size	2000
Number of Islands	1, 5, 10
Objective Functions	f_1, f_1f_2, f_1f_3, $f_1f_2f_3$
Migration Topologies	Random (R), Archive-based (A)
Proportion of Migrating Individuals	0.035
Number of Generations	800
Reproduction Scheme	$\mu + \lambda$
Selection Mechanism	NSGA-II selection
Initialization Method	Half Full, Half Grow
Crossover Probability p_c	0.5
Mutation Probability p_m	0.5
Crossover	One-point, Leaf-biased ($p_{\text{leaf}} = 0.9$) (chosen at random)
Mutation	Uniform, Insert, Shrink, Node Replacement (chosen at random)
Max. Tree Length	30

6.3 Results and Analysis

Selection Procedure

In the following, we provide a comprehensive analysis of the experimental results. Tab. 6.3 gives an overview of the number of successfully solved runs out of 31 runs. To this end, we first determined the best solution per run as the individual in the HOF with the lowest RMSE on the test dataset. A run was counted as solved when the best individual of a run had an error objective $f_1 < 1e-05$ on the test dataset.

6.3.1 Influence of Objective Functions

Objective $f_1f_2f_3$ was most successful in solving Eq. 6.1, across all IM configurations.

For Eq. 6.1, the single-objective single-population algorithm solved five runs successfully. The single-objective IM algorithm solved slightly more runs successfully, when the individuals were distributed over ten islands. A similar behavior could be observed for the objectives f_1f_2. Conversely, the objectives f_1f_3 achieved a higher success rate when only five islands are employed. The objective with the highest success rate was $f_1f_2f_3$, where no clear difference between five or ten islands could be observed. The overall most successful configuration solved 25 runs.

Objective $f_1f_2f_3$ had the highest success rates on Eq. 6.2, with no clearly best performing IM configuration.

Eq. 6.2 was more complex and involved a larger number of computations. This was also reflected in the success rate of the single-objective single-population algorithm, which did not solve a single run successfully. Objective f_1f_2 achieved slightly higher success rates, and $f_1f_2f_3$ solved seven runs with the single-population algorithm and 13 with an IM algorithm. $f_1f_2f_3$ tended to perform slightly better, when the population was distributed over ten islands. Objective f_1f_3 did not improve the success rate compared to the single-objective algorithm. Generally, at least one IM configuration solved more runs successfully compared to the single-population approach of the same objectives. However, no clear best performing IM configuration could be identified [225].

6.3.2 Relationship Between IMGP and Objective Functions

Statistical Comparison of Objective Functions and IMGP Configurations

To validate our observations, we conducted statistical tests on the error distributions of the algorithm variants. For each row, i.e., algorithm variant, the best performing objective in terms of RMSE was the baseline method and was marked in bold (see Tab. 6.4). For each column, i.e., objective, and benchmark instance, the single-population algorithm as a baseline was compared to

Table 6.3: Counts of successful runs for the two benchmark instances and combinations of objectives and IM configurations. A run is successful, when the error f_1 of the best performing individual of the final population is below $1e-05$ on the test dataset [225].

Eq.	# Isl.	Top.	m_{sched}	f_1	f_1f_2	f_1f_3	$f_1f_2f_3$
6.1	1	S	-	5	11	11	15
	5	A	D0	4	17	12	22
		A	D1	4	14	12	19
		R	D0	4	16	12	25
		R	D1	4	14	13	20
	10	A	D0	7	21	5	21
		A	D1	6	21	5	21
		R	D0	6	21	7	23
		R	D1	6	20	7	23
6.2	1	S	-	0	1	0	7
	5	A	D0	0	1	0	11
		A	D1	0	1	0	8
		R	D0	0	3	0	10
		R	D1	0	1	1	7
	10	A	D0	0	2	0	13
		A	D1	0	1	0	13
		R	D0	0	2	0	10
		R	D1	0	2	0	10

Eq.	# Isl.	Top.	m_{sched}	f_1	f_1f_2	f_1f_3	$f_1f_2f_3$
6.1	1	S	-	6.191e-03 ± 5.682e-03	2.777e-03 ± 2.510e-03	**2.559e-03** ± 3.113e-03	**1.152e-03**± 1.593e-03
	5	A	D0	5.921e-03 ± 4.178e-03	**1.378e-03** ± 1.890e-03 ∗	3.781e-03 ± 3.894e-03	**2.822e-04** ± 5.693e-04
		A	D1	5.042e-03 ± 3.483e-03	**1.838e-03** ± 2.214e-03	3.457e-03 ± 3.401e-03	**6.255e-04** ± 1.761e-03
		R	D0	5.723e-03 ± 4.495e-03	**1.488e-03** ± 1.921e-03 ∗	3.732e-03 ± 4.215e-03	**3.059e-04** ± 8.645e-04 ∗
		R	D1	5.443e-03 ± 3.924e-03	**1.373e-03** ± 1.763e-03 ∗	3.274e-03 ± 3.435e-03	**4.157e-04** ± 7.387e-04
	10	A	D0	3.900e-03 ± 4.058e-03	**7.297e-04** ± 1.282e-03 ∗	4.990e-03 ± 3.273e-03	**3.487e-04** ± 8.653e-04
		A	D1	4.188e-03 ± 3.931e-03	**7.466e-04** ± 1.596e-03 ∗	5.378e-03 ± 3.500e-03	**3.317e-04** ± 8.716e-04
		R	D0	5.233e-03 ± 4.577e-03	**6.842e-04** ± 1.233e-03 ∗	3.902e-03 ± 3.332e-03	**4.110e-04** ± 1.114e-03
		R	D1	5.159e-03 ± 4.447e-03	**7.869e-04** ± 1.396e-03 ∗	4.568e-03 ± 3.615e-03	**4.647e-04** ± 1.126e-03
6.2	1	S	-	3.830e-03 ± 2.390e-03	**8.796e-04** ± 8.122e-04	4.201e-03 ± 2.637e-03	**7.766e-04** ± 8.280e-04
	5	A	D0	3.366e-03 ± 1.469e-03	6.993e-04 ± 6.384e-04	3.535e-03 ± 1.816e-03	**2.970e-04** ± 3.448e-04 ∗
		A	D1	3.006e-03 ± 1.725e-03	7.466e-04 ± 6.226e-04	3.270e-03 ± 2.160e-03	**2.494e-04** ± 2.655e-04 ∗
		R	D0	3.544e-03 ± 1.847e-03	9.281e-04 ± 7.806e-04	2.735e-03 ± 1.658e-03 ∗	**3.510e-04** ± 5.497e-04 ∗
		R	D1	3.415e-03 ± 1.671e-03	1.170e-03 ± 9.011e-04	2.785e-03 ± 1.527e-03 ∗	**4.153e-04** ± 4.655e-04
	10	A	D0	3.468e-03 ± 2.015e-03	6.298e-04 ± 5.999e-04	3.417e-03 ± 1.608e-03	**2.658e-04** ± 6.519e-04 ∗
		A	D1	3.288e-03 ± 1.980e-03	7.637e-04 ± 7.666e-04	3.343e-03 ± 1.706e-03	**2.691e-04** ± 3.139e-04 ∗
		R	D0	3.146e-03 ± 1.514e-03	6.788e-04 ± 5.419e-04	3.224e-03 ± 1.592e-03	**3.540e-04** ± 3.907e-04 ∗
		R	D1	3.113e-03 ± 1.671e-03	7.892e-04 ± 7.575e-04	3.165e-03 ± 1.536e-03	**2.796e-04** ± 3.234e-04 ∗

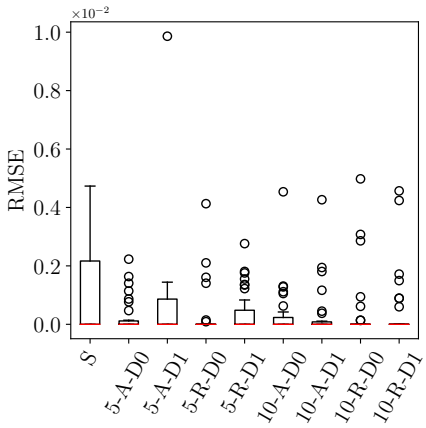
Table 6.4: RMSE values (mean ± standard deviation) over 31 runs. Numbers in bold indicate the best result for each row, i.e., which objective performed best for the specific IM configuration. The asterisk symbol indicates that an IM configuration outperforms the respective single-population variant of the same objectives and benchmark instance with statistical significance [225].

all IM variants. The asterisk symbol indicates that an IM configuration performed significantly better than the respective single-population algorithm on this benchmark instance. We used the one-sided Mann-Whitney-U statistical test for each algorithm to test whether its performance was significantly worse than the baseline method. The level of significance was $\alpha = 0.05$. The results are displayed in Tab. 6.4.

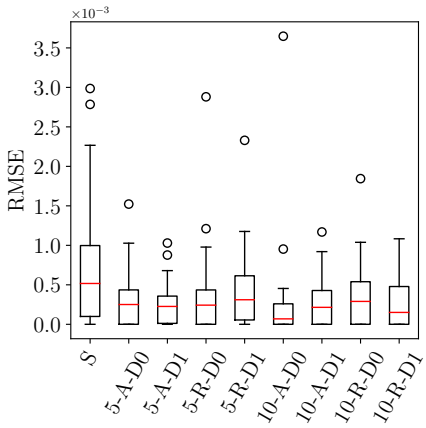
On Eq. 6.1, the best performing objectives for the single-population algorithm were f_1f_3 and $f_1f_2f_3$. The IM variants performed best when f_1f_2 or $f_1f_2f_3$ were used, which outperformed the other two objectives but were not significantly different from each other. Interestingly, most IM variants outperformed the single-population algorithm when f_1f_2 was used. However, only one IM algorithm resulted in a better performance when the dimension penalty was added as an objective, i.e., when $f_1f_2f_3$ was employed. Taking a look at the results of Eq. 6.2, the objective $f_1f_2f_3$ outperformed the other objectives for all IM settings, except for the single-population algorithm. Furthermore, almost all IM configurations outperformed the single-population algorithm when $f_1f_2f_3$ was used as an objective. For the other three objective variants, most IM

Figure 6.2: Distribution of best RMSE values over 31 runs for the single-population GP and all IMGP configurations.

(a) Archive-based IMGP on Eq. 6.1 using objectives $f_1 f_2 f_3$.



(b) Archive-based IMGP on Eq. 6.2 using objectives $f_1 f_2 f_3$.



algorithms did not show a significantly better result than the single-population algorithm. [225]

The three objectives variant $f_1 f_2 f_3$ with IMGP performed significantly better than single-population on Eq. 6.2, but not on Eq. 6.1.

It was noteworthy that the variant using three objectives and IM significantly outperformed the single-population algorithm for almost all configurations on Eq. 6.2, but not on the simpler Eq. 6.1. To gain further insights into the underlying RMSE distributions of the three objective experiments, Fig. 6.2 presents the respective boxplots for both benchmark equations. Fig. 6.2a clearly shows that the median values of all experiment variants, including the single-population one, had a median value of zero. Furthermore, we could observe a certain number of outliers in the IM variants, which altogether supports the null hypothesis of no statistically significant differences in pairwise comparison between the single-population and IMGP variants. In Fig. 6.2b, larger differences in the median values are apparent, which is also reflected in the results of the statistical analysis in Tab. 6.4 [225].

No significant differences were observed between the IM variants.

In addition to the results in Tab. 6.4, we performed a Kruskal-Wallis test on the IM variants for each objective and benchmark instance. No significant differences between the IM variants could be identified for all objectives and benchmark instances. While Tab. 6.3 indicates a tendency that ten islands perform better than five for some algorithm variants, this assumption could not be supported in the statistical tests. The effects of the migration topology and the distribution of migrations over the algorithm runtime were negligible.

Migration increased the success rate, almost independent of the IM variants.

Overall, we could observe for the less complex Eq. 6.1, that the IM algorithm only showed significantly better results when the objective $f_1 f_2$ was utilized. The more complex Eq. 6.2 benefited from both, the use of a multi-objective algorithm $f_1 f_2 f_3$ and the distribution of the population on multiple islands. In cases where the results improved through the implementation of an IM algorithm, this improvement occurred regardless of the specific choice of IM hyperparameters. We can conclude that within the considered hyperparameter space, migration had a positive impact on the success rate, almost independent of the exact IM configuration. We would like to point out that this statement is only valid within the scope of our experiments, using a limited number of hyperparameters that have been proved useful in previous studies. We did not test extreme cases here, such as migrating individuals in every generation or exchanging a larger number of individuals, which might have behaved differently.

Migration reduced the variability in the final result when an algorithm is repeated multiple times.

In addition to the success rate and statistical comparison of the final results, we were also interested in the variability, or spread, with regard to the final results when the same algorithm was repeated multiple times. Fig. 6.3 displays boxplots with the RMSE distributions of selected single-population GP and IMGP variants over 31 runs. We were mainly interested in the interquartile

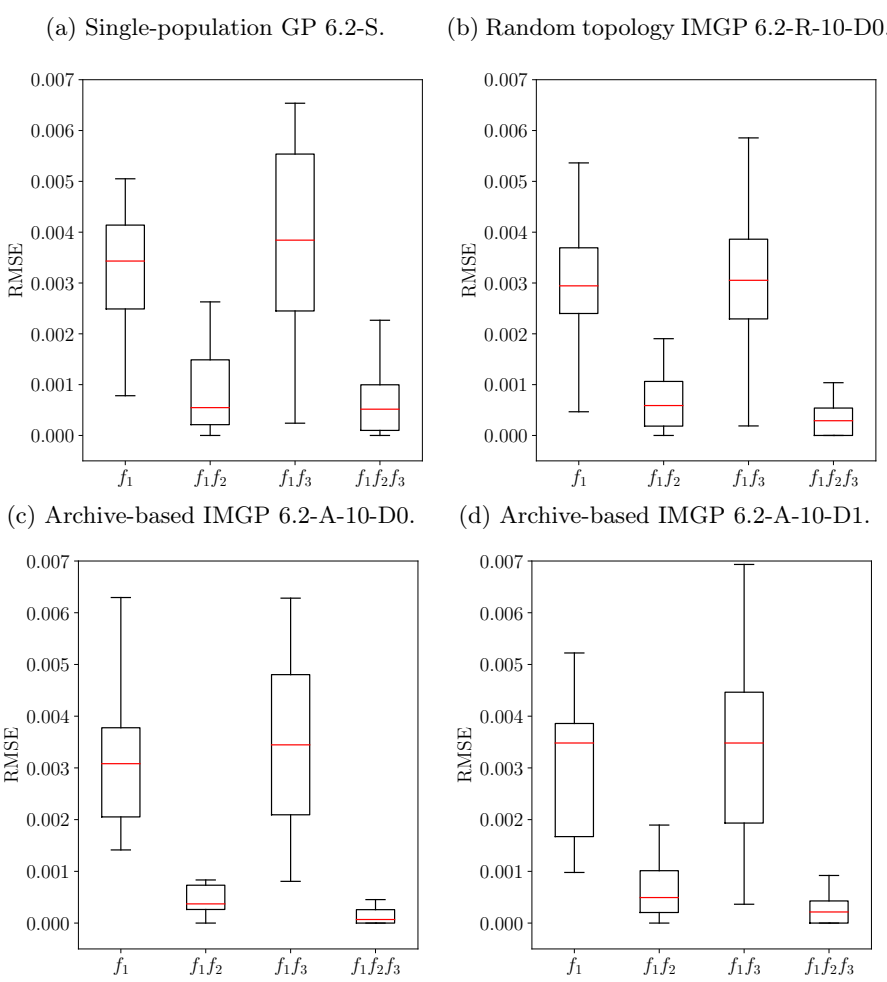


Figure 6.3: Distribution of best RMSE values over 31 runs for selected IMGP configurations on Eq. 6.2.

ranges of the boxplots, as indicated by the upper and lower bounds of the black boxes around the median RMSE displayed in red. Compared to the single-population GP in Fig. 6.3a, the IMGP variants consistently exhibited lower spread in their final results. We could also observe an effect of the objective functions on the spread in the final results: when the median RMSE is larger, the spread is also larger, as for objectives f_1 and f_1f_3.

Considerations on Choice of Algorithm Parameters

The extent the proposed methods could help to discover new equations from data in the future is of great interest. We assume that new equations are likely to involve a high number of operations and therefore closer to the difficulty of Eq. 6.2. Preferring RMSE over MaxAE as the first objective could help to discover good equations even on noisy data. Based on the results of this chapter, we strongly recommend using correlation as a supporting objective f_2. The satisfaction of physical laws is required for most engineering applications, which is why we suggest including f_3. We should point out that our benchmark equations did not contain unknown coefficients, which will likely be the case when discovering new equations. Since coefficient units are not known beforehand, an adapted dimension penalty is required, which takes this into account. Altogether, we recommend using the objectives $f_1f_2f_3$ as well as an IM configuration for such GP algorithms, especially since the IM comes at no additional computational cost compared to the single-population algorithm [225].

6.4 Summary

This chapter aimed at enhancing the success rate of GP algorithms for symbolic regression. We presented a comparison of different objective functions and

configurations of island model algorithms. The objectives comprised an error function, a correlation coefficient, and a dimension penalty. We tested the performance of these objectives on a varying number of islands, two migration topologies and two distributions of the migrations over the algorithm runtime. The 36 algorithm variants were tasked with solving two benchmark equations from the fluid mechanics area, which previously showed high variations between the results of multiple runs. The results of our experiments showed a strong influence of the objective function on the success rate of the algorithm. For some objectives, the results improved further when an island model approach was used, compared to a single-population algorithm. These objectives were also the best performing objectives overall. No significant differences were found between the IM configurations themselves, suggesting that results for some targets improve as long as migration is performed.

Our results provide a promising starting point for future research directions. Our proposed algorithms can be applied to a more diverse set of benchmark equations to further verify our recommendation for discovery of new equations. In addition, approaches to account for unknown units of the coefficients during dimensional analysis form an important extension of this chapter.

7 Graph Neural Networks as Inductive Bias for Genetic Programming

Introduction and Chapter Goals

As outlined in Chapters 4 and 5, the complexity of the problems limited the applicability of GP and the quality of the final solutions. This chapter focuses on the use of *message passing neural network*s (MPNNs) as an intermediate step towards symbolic models, as they provide a well-motivated inductive bias for both problems from robotics and fluid mechanics. They share similar characteristics, which allow modeling the problems as graphs of entities with interactions between them. Our goal for this chapter is to demonstrate this modeling process including further domain knowledge, and assess how well both problems can be learned with MPNNs. For the fluid mechanics problem, we will furthermore take a look at the underlying equations that replace the network block in the MPNN.

7.1 Modeling of Particle-Laden Flows

The following section is largely based on the author's publication [223].

From Velocity Fields to Force Prediction in Complex Flows

In Chapter 5, we discovered that the applicability of GP to predict velocity fields for more than two particles faces limitations, as these fields resemble a continuous velocity distribution across the flow domain. Consequently, a more promising approach may be to shift the focus to directly predicting the hydrodynamic forces $\mathbf{F}$ acting on individual particles. Accurately estimating these forces remains a challenging, open problem in the literature, as discussed in Sec. 5.1. Various approaches have been presented to approximate the value of $\mathbf{F}$, including empirical models [228, 268], the PIEP model [2, 16] and PINNs [255]. While empirical approaches can only predict the mean force, other methods show promise in predicting variations from the average. However, these methods often lack explainability and predict the force $\mathbf{F}$ with some error. Building upon the works of the authors presented in Chapter 5, Seyed-Ahmadi et al. [255] and Cranmer et al. [60, 162], the objective here is to develop interpretable symbolic models for $\mathbf{F}$, which can capture the complex interactions for a large number of particles. Interpretable models for this problem are desirable as they allow for deeper understanding and analysis of the underlying interactions, which have remained opaque in previous approaches.

The Overall Learning Pipeline

As depicted in Fig. 7.1, the proposed learning pipeline comprises two phases: In the first phase, an MPNN is trained on high-resolution input data. The inductive bias of the MPNN determines the internal structure, i.e., which particles

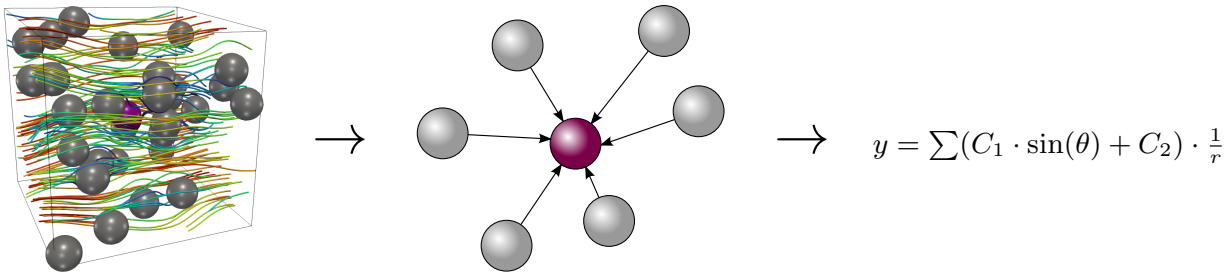


Figure 7.1: Symbolic models are generated from simulation data using an MPNN as a surrogate model. Physical particles in the simulation translate to nodes in the MPNN [223].

interact with others and how the influences of multiple neighboring particles are aggregated. This surrogate model facilitates the development of symbolic models, since the underlying shape of the equation is partially determined beforehand. Subsequently, the GP algorithm fits symbolic models to the output of the internal structures of the MPNN, rather than the actual target variable. The prediction of the target variable is achieved by aggregating the outputs of the symbolic models over all neighboring particles, using the same aggregation scheme as previously employed in the MPNN [223].

7.1.1 Translating Interacting Particles to Graph Structures

Particles translate to nodes, interactions to edges in the MPNN.

Many systems in science and engineering applications can be represented by graphs, such as spring systems [60], the solar system [162], or particles in a particle-laden flow. This supported the use of MPNNs to model interactions between objects or particles, as for our case study. MPNNs are a subtype of *graph neural network*s (GNNs) (see Sec. 2.2.2). They contain network models for each internal structure of a graph. In our model, a particle translates to a node in the MPNN, which is described by the node model Φ^n. A system of q particles is represented by a graph with q nodes n_i, where $i = 1 \dots q$. Interactions between particles are represented by edges between nodes and described by the edge model Φ^e. A node n_i has incoming and/or outgoing edges from/to the nodes in its neighborhood $\mathcal{N}_i$. The message function $m_{i,j}$ captures pairwise interactions between two nodes n_i and n_j, where $n_j \in \mathcal{N}_i$. The neighborhood can be defined by a number of closest nodes to n_i or all nodes within a certain distance from n_i. The pairwise interaction is influenced by the current state of the interacting particles, so that the input to the edge model comprises the features of the two interacting particles. The node model updates the state of a particle as a function of the current state of a particle n_i, as well as the aggregated incoming edge messages. Both Φ^e and Φ^n use shared parameters for all pairwise interactions and node updates, and are updated according to Eqs. 2.21−2.23 [223].

Underlying Model Structures Suitable for Flow Prediction

An MPNN facilitates different ways of predicting a target variable, i.e., different underlying structures. Since the optimal structure of the model to predict a target variable y is unknown, our framework proposed two variants aligned with the structure of the problem at hand:

1. $y = g(x) = \sum_{j \in \mathcal{N}_i} m_{i,j}$: Only the edge model Φ^e is captured. The target variable is the sum of the edge messages received by a node.
2. $y = f(g(x)) = f(\sum_{j \in \mathcal{N}_i} m_{i,j})$: Both the edge model Φ^e and node model Φ^n are captured. The target variable is a function of the aggregated edge messages. Thus, the aggregated edge messages are an input to the node model, which predicts the target variable.

Fig. 7.2 depicts these two variants using the example of a particle-laden flow. The internal structure of an MPNN is separable, which meant that we could

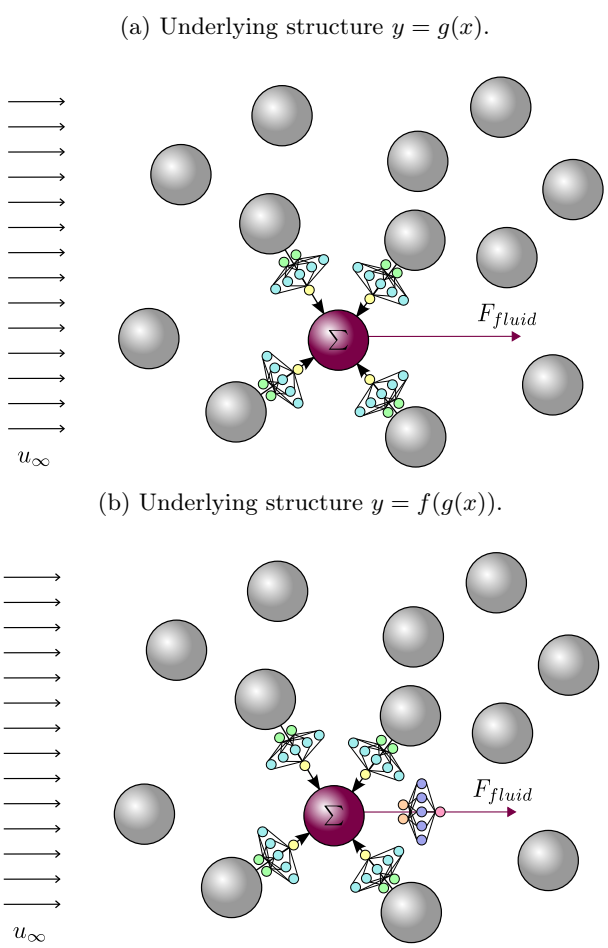


Figure 7.2: MPNN to predict $\mathbf{F}$ *imposed on the red particle in a particle-laden flow, given four particles in its neighborhood* $\mathcal{N}_i$*.* u_∞ *is the undisturbed flow velocity [223].*

fit separate symbolic models to the outputs of the edge and node models. This greatly facilitated the equation fitting in the next step.

7.1.2 Replacing Network Blocks with Symbolic Models

Advancements in Symbolic Regression for Scientific Equations

We employed a GP algorithm to develop symbolic models from the output values of the node and edge models of the MPNN. With growing interest in symbolic regression for engineering and scientific applications, the basic GP algorithm has been enhanced with techniques from the area of machine learning. An important property is the possibility to include and fit constants in the equations, usually achieved by a regression algorithm on top of the evolution of equations. Other techniques are batch-wise training to process big datasets in a reasonable amount of time, and the use of sophisticated error functions [223].

Genetic Programming Algorithm

Problem-Specific Design Choices

The proposed GP algorithm made use of the training features, which could include raw data as well as pre-processed or transformed data to induce prior knowledge about the problem. In addition, the algorithm employs constants, which were fitted through a regression algorithm. Since no ground truth equation was available, selecting an appropriate function set was a complex task that significantly influenced the results. Preliminary experiments and a coarse function tuning had shown that the set of functions and operators $\{+, *, \sin(\circ), \cos(\circ), \tan(\circ), e^{(\circ)}, \log(\circ)\}$ yielded satisfactory results. The fitness function to be minimized was the commonly used *mean squared error* (MSE).

Adaptations to Different Underlying Structures

Depending on the underlying structures imposed by the MPNN, the GP algorithms slightly differed [223]:

1. $y = g(x) = \sum_{j \in \mathcal{N}_i} m_{i,j}$: The symbolic model $\Phi^{e'}$ replaced the message

model Φ^e, and was thus fitted to the output of the message function, which was recorded during MPNN training. Constants in the resulting equations were then refitted to the original target variable to avoid the accumulated approximation error. To this end, we employed the Levenberg-Marquardt algorithm and used the constants found by the GP algorithm as starting values.

2. $y = f(g(x)) = f(\sum_{j \in \mathcal{N}_i} m_{i,j})$: The first symbolic model $\Phi^{e'}$ replaced the message model Φ^e and followed the same procedure as in (1). The second symbolic model $\Phi^{n'}$ replaced the node model Φ^n and predicted the target variable, given the influence of the neighboring particles. Thus, the node model received the aggregated pairwise interactions $\sum_{j \in \mathcal{N}_i} m'_{i,j}$, computed by the symbolic model $\Phi^{e'}$, as an additional input feature. To refit the constants, the inner function $\Phi^{e'}$ was plugged into the outer function $\Phi^{n'}$.

Techniques for Physically Meaningful Equations

Reconciling Model Simplicity, Unit-Awareness and Constant Fitting

Physical laws often follow relatively simple equations. Thus, our GP algorithm aimed at finding equations of low complexity. At the same time, these equations should be in line with physical laws in terms of units. While unit-aware GP approaches like grammar-based GP or a dimension penalty as an additional objective are often effective to avoid unit violations, they are complex to implement and sometimes restrict the search space in an undesirable way. Currently, there is no GP framework publicly available that allows both constant fitting and unit awareness. Since the fitting of constants has a high influence on the numerical accuracy of the equations, we decided to use the *PySR* framework [58] that offers constant fitting. Moreover, recent research indicated that also relatively simple techniques can yield satisfactory results [60]. To this end, our algorithm employed a complexity measure, complexity-constrained function inputs, as well as certain building rules for the parse trees [223].

Higher Complexity Values for Constants and Non-Linear Operations

Complexity measure: To compute the complexity of an equation, each operation, function, feature, and constant were assigned a complexity value. The total equation complexity is the sum of the complexity values of the used primitives. The complexity values were determined by a coarse hyperparameter tuning, with complexity values up to 4 for nonlinear operations. Dividing the set of primitives into two groups, i.e., less and more complex with associated complexity values of 1 and 2, gave the most satisfactory results in terms of accuracy and interpretability.

Binary operators like addition, subtraction, and multiplication, as well as the training features were assigned a complexity value of 1. Constants play an important role in numerous physical laws, such as the gravity constant, to name one example. When the number of constants in an equation is unknown, using too many of them in the same expression can lead to overfitting the training data. Thus, we assigned a higher complexity value of 2 to constants. Unary functions such as $\sin(\circ)$, $\cos(\circ)$, $\tan(\circ)$, $e^{(\circ)}$ and $\log(\circ)$ apply a non-linear transformation to the input. Consequently, they are associated with a higher complexity value of 2 compared to the basic operators. The unary operation $\frac{1}{\circ}$ was associated with a complexity of 1 [223].

Controlling Model Parsimony by Limiting Nested Functions

Complexity-constrained function inputs: Another technique to keep the expressions simple yet effective is to restrict the input of certain operations to a maximum allowed complexity. Our algorithm restricted the input complexity of trigonometric, logarithmic, and exponential functions to 8. This meant, an expression like $y = \sin(2.0 \cdot x - 3.0)$ with an input complexity of 7 were allowed. However, $y = \sin(2.0 \cdot x + \log(x) + 3.0)$ with an input complexity of 11 exceeded the limit [223].

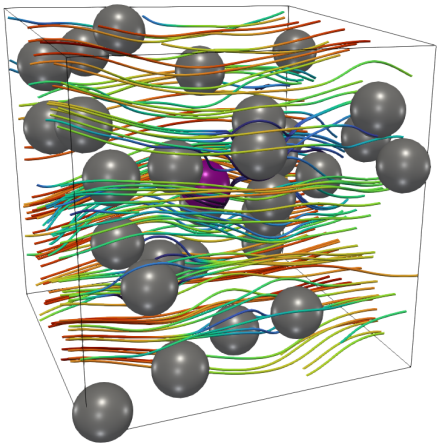


Figure 7.3: Random array of stationary spherical particles at $\phi = 0.064$ [223].

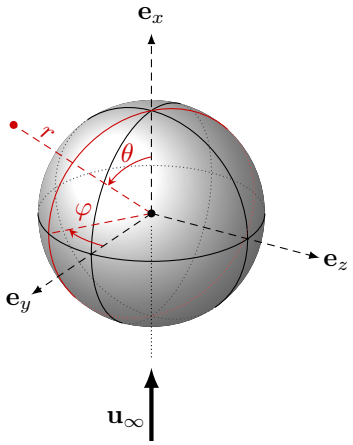


Figure 7.4: Spherical coordinate system with radius r, polar angle θ, and azimuthal angle φ [223].

Control Nesting of a Single Operation

Building rules: Preliminary experiments indicated that GP algorithms sometimes tended to include multiple nested functions in expressions, for instance $\sin(\cos(\sin(\circ)))$. This behavior is to be avoided, as it can lead to the model overfitting the training data and usually has little meaning in terms of explainability. Consequently, we limited the nesting of trigonometric functions to a maximum of 1, so that $\sin(\sin(\circ))$ was allowed, but further nesting with any trigonometric function was prohibited [223].

7.1.3 Data Generation

Simulating Stokes Flow with the Method of Regularized Stokeslets

We considered the flow past a stationary array of monodisperse spherical particles in the Stokes regime ($\mathrm{Re} \to 0$), at which the viscous forces dominate. In this regime, the flow is governed by the Stokes equations in Eqs. 5.1 and 5.2. There is no closed-form solution for such equations in complex configurations that include more than a single spherical particle. However, a solution can be built from the superposition of fundamental solutions due to the linearity of the governing equations. The method of Regularized Stokeslets [56] was applied to construct the solution of the flow around the array of particles. A single regularized Stokeslet solves the flow driven by locally distributed force ($\mathbf{F} = \mathbf{g}\phi_\epsilon(|\mathbf{x} - \mathbf{x}_0|)$) in free space, where ϕ_ϵ is an isotropic regularization kernel with compact support over the length ϵ. Each particle is represented by a group of locally distributed forces to achieve the no-slip at the particle surface [223].

Simulation Parameters

A random array of 30 spherical particles was generated in a unit cube except for one particle which is placed at the center of the cube. Each particle was represented by 300 force markers. The free stream flowed in x-direction with uniform velocity $u_\infty = 1\mathrm{m/s}$ and the fluid viscosity is $\mu = 1\ \mathrm{kg/(m\ s)}$. The force $\mathbf{F}$ has three components and was computed on the particle located at the center of the cube. We generated a training dataset consisting of 500 randomly

Table 7.1: Overview of domain knowledge integrated as bias into the MPNN and GP algorithms to approximate the variation from the mean drag in Stokes flow.

Bias Type	Bias
Observational	Conversion of input data from Cartesian to spherical coordinate system
Observational	Augmentation of training data for implicit representation of symmetries in the dataset
Inductive	MPNN as inductive bias to approximate the underlying pairwise interactions between particles and subdivide the problem
Learning	Limiting the amount of nestings per operation
Learning	Higher complexity values for constants and non-linear operations
Learning	Setting an upper limit for the complexity of inputs to trigonometric, logarithmic, and exponential functions

initialized particle arrangements, and a separate test dataset of 500 samples obtained using different random seeds in the simulation. We provided benchmark data for each of the following volume fractions $\phi = [0.064, 0.125, 0.216, 0.343]$. An exemplary array of particles with streamlines of the flow is depicted in Fig. 7.3. Each training sample encompassed the following features [223]:

- Relative positions $\mathbf{r}_i$ of the 29 neighboring particles
- Average fluid velocity $\bar{\mathbf{u}}^f$ within the unit cube
- Streamwise force component F_D exerted by the fluid on center particle

7.1.4 Data Preprocessing

Conversion from Cartesian to Spherical Coordinate System

The raw data generated by the Stokes flow solver from Sec. 7.1.3 underwent further transformations before serving as input to the MPNN. Initially, the raw particle locations were represented in a three-dimensional Cartesian coordinate system as (x, y, z). Preliminary experiments had indicated that the GP algorithms perform better when locations are available in spherical coordinates. Results of a related research paper modeling the orbital mechanics of planets in the solar system employed a similar transformation of relative locations [162]. The MPNN performance remained similar for both configurations. Thus, we converted the particle locations to spherical coordinates r, θ and φ (see Fig. 7.4 for the exact definition). We assumed that it would behave this way because the relative particle distance r played an important role in the underlying symbolic model. Furthermore, the trigonometric functions employed in the function set of the GP algorithm were more meaningful with an angle like θ and φ as input. This saved the algorithm an intermediate step in computing a dimensionless quantity from the features in Cartesian coordinates.

Data Augmentation Procedure

To increase the number of available data samples, we augmented the data by rotations around the axis of the free stream. For each particle arrangement, seven consecutive rotations by $\frac{\pi}{4}$ each were performed, so that a total of eight configurations were attained per arrangement. This provided the added benefit of representing symmetries around the free flow direction in our data. We split the data with a 3:1 ration into training and validation sets [223]. After augmentation and splitting, a total of 3,000 samples per volume fraction was available for training.

7.1.5 Experiment Design

Focus on Prediction of the Deviations from the Mean Drag Force

To examine the general applicability of our approach, we predicted the streamwise force component, i.e., the drag force. Once this concept had been proven, it could be applied to the other force components, lift and torque, as well.

Since the mean force $\langle F_\mathrm{D} \rangle$ could already be approximated from existing correlations [268], we could predict the deviation from the mean force. In general, this can have the same order of magnitude as $\langle F_\mathrm{D} \rangle$ itself.

Input Features to the Edge and Node Models

We investigated the viability of the presented approach using benchmark data from the Stokes flow (i.e., $\mathrm{Re} = 0$), with four different particle-volume fractions ϕ. For each dataset, separate models were trained for the underlying structures $y = g(x)$ and $y = f(g(x))$. The features of the neighboring particles, i.e., the input features to the edge model, were the relative position from the center particle in spherical coordinates r, θ, φ. The training features of the particle of interest comprised the local average velocity in x, y and z direction, $\bar{u}_x^f$, $\bar{u}_y^f$ and $\bar{u}_z^f$ [223].

Network Parameters of the Edge and Node Models

The edge and node models of the MPNN comprised two fully connected hidden layers with 30 neurons each. We used the hyperbolic tangent as nonlinearity. For both $y = g(x)$ and $y = f(g(x))$ as underlying structures, the output of the edge model was recorded during the training of the MPNN to be used as target features of the GP algorithm. The learning rate with an initial value of 0.002 was adjusted during the training process. The Adam optimizer optimized the model parameters. We trained the model for 5000 epochs to minimize the MSE as loss function. The MPNN was implemented using *PyTorch Geometric* [85].

Algorithm Parameters of GP

We ran the GP algorithm for 200 iterations, with a population size of 100 individuals. The multi-objective algorithm minimizes the MSE as well as the complexity value of an equation. The best individual from the final Pareto front was identified using a combined measure of accuracy and complexity, as implemented in [57]. The algorithm employed the problem-specific parameters as described in Sec. 7.1.2, and used the standard configuration of *PySR* with regard to genetic operators and operator probabilities [223].

Considerations on Complexity of Constants

In the first trials, we observed that the nested symbolic models $y = f(g(x))$ were more complex than $y = g(x)$, with a tendency to mainly use constants in the outer equations. This can be explained by the two consecutive GP runs for f and g. To keep the comparison fair, we wanted to allow the algorithm for $y = g(x)$ to use more constants, by reducing the constant complexity to 1. Since the structure of an accurate equation was unknown, and fewer constants could be beneficial for generalization, we still ran experiments for $y = g(x)$ and a constant complexity of 2. Considering the four benchmark datasets, this made a total of twelve experiment instances. The training data and code for the fluid mechanics problem described in this chapter are publicly available at https://github.com/juliareuter/flowinGN. To summarize the proposed algorithms and problem-specific settings, Tab. 7.1 gives an overview of the different types of domain knowledge that were integrated along the learning pipeline [223].

7.1.6 Results and Analysis

MPNN Performance and Model Selection as Input for GP

Since the algorithm comprised two steps, we applied the following procedure: The MPNN was trained ten times for each experiment variant. We observed similar accuracies for all runs, which were comparable to those of state-of-the-art-approaches [16, 255]. We randomly selected one of the ten models as our basis model. In the next step, we employed the GP algorithm to replace this basis model with symbolic models. For statistical comparison, we performed 31 independent realizations of the GP algorithm per experiment instance. The experiments were analyzed in terms of the overall algorithm performance, the explainability of the resulting equations, as well as tests on unseen data from different realizations of the simulation.

Overall Algorithm Performance

Statistical Test on Distributions of MSE Values

Fig. 7.5 displays the MSE distributions over 31 realizations for each experiment variant. We used the Holm-Bonferroni test to compare the results for each ϕ. The best variants are displayed in bold. For $\phi = 0.064$, no statistically signif-

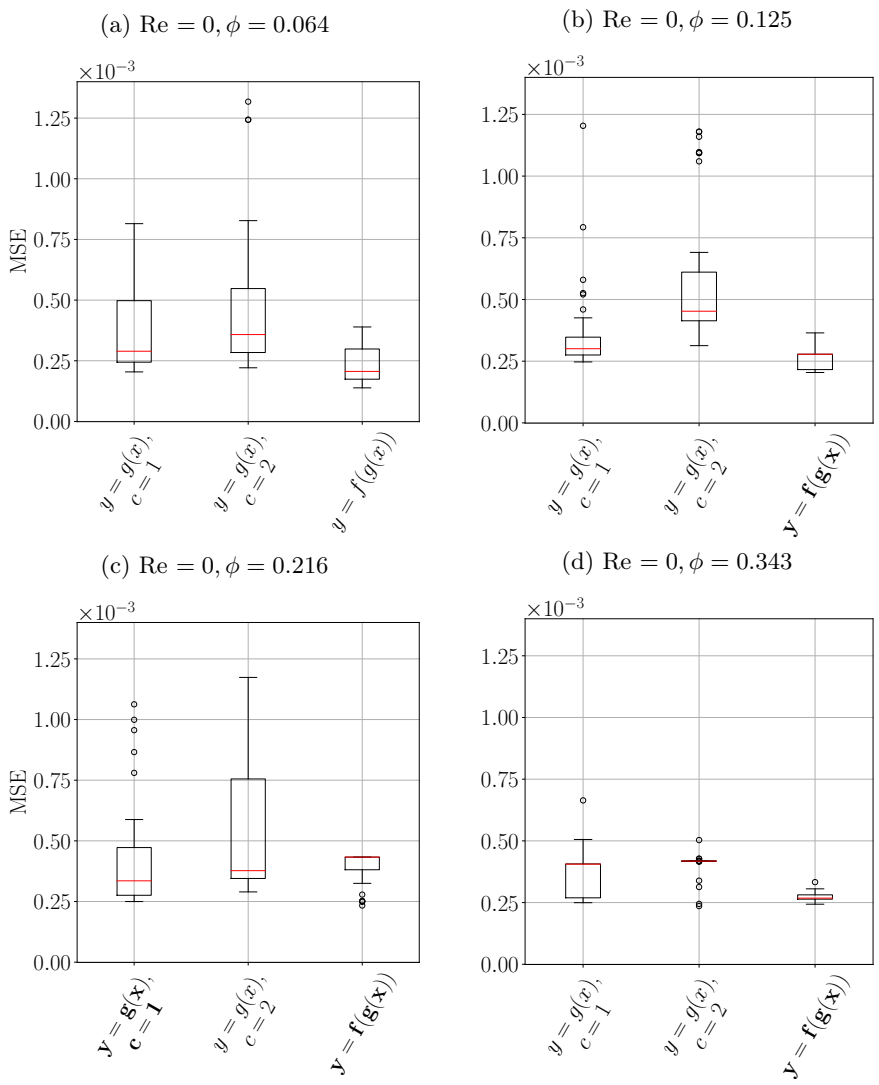


Figure 7.5: MSE for different experiment instances over 31 realizations. The variable c indicates the constant complexity for $y = g(x)$. Bold experiments performed best [223].

icant difference between the three variants was identified. For $\phi = 0.125$ and $\phi = 0.343$, the nested function performed better than the two other variants. The variant $y = g(x)$ with a constant complexity of 1 and the nested function performed best for $\phi = 0.216$.

$y = f(g(x))$ produced a lower spread in MSE compared to $y = g(x)$.

We can observe that all experiment variants for all ϕ achieved MSE values of similar magnitude. In general, the nested function $y = f(g(x))$ had a lower spread compared to $y = g(x)$ over 31 runs, i.e., was more reliable to achieve good results. While the medians differed, the best models found by each algorithm had almost the same error value. For most ϕ, no significant difference between the constant complexities $c = 2$ and $c = 1$ for $y = g(x)$ was observable [223].

Explainability of Equations

The resulting equations mainly used the features r and θ.

For a more profound analysis of the resulting equations, we selected the best and/or most frequently found symbolic model for each experiment variant. The constants of these equations were refitted to the original dataset, since they had been trained on the outputs of the MPNN edge model rather than the target variable. Tab. 7.2 shows the refitted equations together with their MSE values on the test dataset. For comparison, the MSE of the MPNN on the same dataset is displayed [223]. The equations were concise across all experiment instances. It became obvious that the algorithm settings successfully prevented function nesting as well as complex input arguments for trigonometric, logarithmic, and exponential functions. Almost all equations were physically meaningful without

ϕ	Experiment	Equation	GP MSE	GN MSE
0.064	$y = g(x)$, $c = 2$	$\sum\left(0.01146r + 0.01146\sin(\theta) - 0.0142\right)\frac{1}{r}$	0.000209	0.000120
	$y = g(x)$, $c = 1$	$\sum\left((0.03448r + 0.03448\sin(\theta) - 0.04238)(-\log(r))\right)$	0.000188	0.000120
	$y = f(g(x))$	$0.0992\sum\left((r(\sin(\theta) - 0.1312) - 0.1983)(-\log(r))\right) + \bar{u}_x^f - 0.3177$	0.000157	0.000106
0.125	$y = g(x)$, $c = 2$	$\sum\left(0.01397\sin(r) + 0.01397\sin(\theta) - 0.01724\right)\frac{1}{r}$	0.000284	0.000173
	$y = g(x)$, $c = 1$	$\sum\left(0.00839 + (0.01578\sin(\theta) - 0.01644)\frac{1}{r}\right)$	0.000260	0.000173
	$y = f(g(x))$	$0.0597\sum\left((\sin(\theta) - 0.45368 - \frac{0.12479}{r})e^{-r}\right) - 0.0616$	0.000209	0.000146
0.216	$y = g(x)$, $c = 2$	$\sum\bar{u}_x^f(\sin(\theta) - 0.57328 - \frac{0.10557}{r})$	0.000316	0.000206
	$y = g(x)$, $c = 1$	$\sum\left(0.00944 + (0.01932\sin(\theta) - 0.01982)\frac{1}{r}\right)$	0.000247	0.000206
	$y = f(g(x))$	$0.1166\sum\left((0.17448\sin(\theta) - 0.08318 - \frac{0.01419}{r})\frac{1}{r}\right) - 0.1602$	0.000248	0.000167
0.343	$y = g(x)$, $c = 2$	$\sum\left((0.08249\sin(\theta) - 0.07348)(-\log(r)) + 0.00539\right)$	0.000239	0.000191
	$y = g(x)$, $c = 1$	$\sum\left((0.08749\sin(\theta) - 0.07348)(-\log(r)) + 0.00423\right)$	0.000239	0.000191
	$y = f(g(x))$	$0.3904\sum\left(\left(0.10982e^{\sin(\theta)} - 0.26635\right)(-\log(r)) + 0.0165\right) - 0.0421$	0.000219	0.000197

Table 7.2: Symbolic models with refitted constants [223].

the use of a dimension penalty or grammar-based approach, only through including prior problem knowledge as constraints. Solely $\sin(r)$ and $\exp(\sin(\theta))$ were unusual terms. However, this method generally cannot guarantee that physics-conformal equations are evolved. While the input comprised the six features r, θ, φ, $\bar{u}_x^f$, $\bar{u}_y^f$, and $\bar{u}_z^f$, mainly r and θ were used, twice as well $\bar{u}_x^f$ came into play [223].

MPNNs generally exhibited slightly lower MSE values compared to the equations obtained with GP.

Taking a look at the MSE values, the GP equations performed slightly worse than the MPNN . The errors of the symbolic models were in the same order of magnitude of 10^{-4} as the MPNN, but were sometimes about 1.5 times higher. The underlying structure $y = f(g(x))$ performed better for all benchmark datasets. The best equations with $y = g(x)$ as the underlying structure demonstrated better performance for $c = 1$ compared to $c = 2$ across all benchmark instances [223].

$y = f(g(x))$ used constants more frequently than $y = g(x)$.

For the underlying structure $y = g(x)$, we examined two complexity values for constants, of $c = 2$ and $c = 1$. The constant complexity $c = 1$ resulted in one additional constant for $\phi = 0.125$ and $\phi = 0.216$. The other instances employed the same number of constants for both complexity values. Comparing the number of constants of the two underlying structures, $y = f(g(x))$ always contained one or two more constants than $y = g(x)$ [223].

All equations shared similar building blocks, comprising the features r and θ.

Similar patterns across equations of all experiment instances could be identified. Each equation contained a building block that accounted for the distance of a neighboring particle: The terms $\frac{1}{r}$, $-\log(r)$ and $\exp(r)$ scale the influence of a neighboring particle on the particle of interest, i.e., they decreased with increasing radius r. Furthermore, each equation contained a larger building block, which included $\sin(\theta)$ and constants or other small terms. We can assume that this building block determined the influence of a particle, which was then scaled with the distance to the center particle. Fig. 7.6 displays the function values of some of the identified building blocks [223].

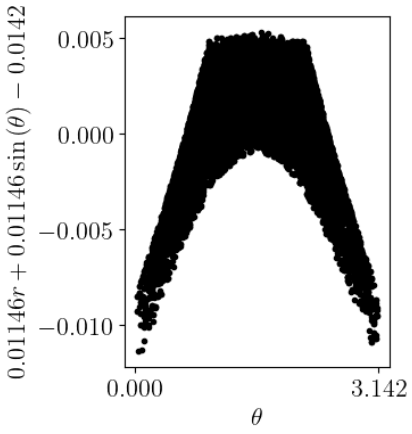
Testing of Symbolic Models

No overfitting of equations could be detected on the test sets with data from a different flow realization.

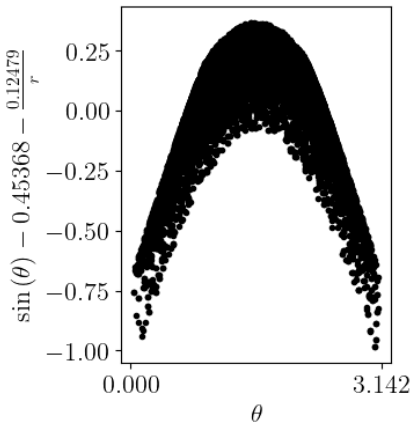
Due to the complex underlying relations, overfitting to training data is a common issue in machine learning for fluid mechanics [16]. Thus, we tested the equations found by the GP algorithm on a dataset with the same values for Re and ϕ, but from a different simulation realization. Fig. 7.7 exemplarily depicts the normalized predictions of the deviation from the mean force $\langle F_\mathrm{D} \rangle$ for $\phi = 0.216$. Fig. 7.7b illustrates the predictions for 500 instances from the

Figure 7.6: Insights into frequently used building blocks in the symbolic models [223].

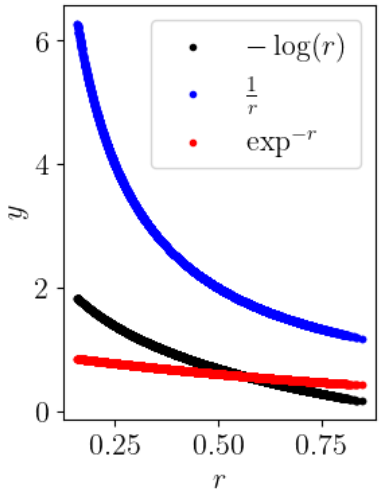
(a) Distribution of functions values over θ as in an equation for $\phi = 0.064$.



(b) Distribution of functions values over θ as in an equation for $\phi = 0.125$.



(c) Decaying building blocks over r often appear in the equations.



same realizations as used in the training data, and Fig. 7.7a from a different realization. The plot, as well as the MSE values of 0.000247 (training) and 0.000225 (test) for $y = g(x)$ and 0.000248 (training) and 0.000226 (test) for $y = f(g(x))$, indicate that the equations identified actual underlying patterns and did not overfit the training data. The other benchmarks revealed a similar behavior [223].

The overall results confirmed the applicability of the proposed approach to approximate the deviations from the mean drag force in a particle-laden flow in the Stokes regime. In the following section, we will assess the potential to learn the inverse kinematics problem of robotic manipulators, which is an application from a different engineering field but shares certain characteristics with the fluid mechanics problem presented here.

7.2 Modeling the Inverse Kinematics Problem

From Cooperative Coevolution to MPNNs for Representation of the Kinematic Chain

In Chapter 4, we applied GP to learn the inverse kinematics of a 5 DOF manipulator, which is considered a non-standard manipulator due to the absence of one joint. We developed separate equations for each joint, predicting the necessary joint angle to reach a target pose. The complex interactions among joints in the kinematic chain were addressed using a cooperative coevolutionary approach, which achieved a position RMSE of 0.0343m in the best case. How-

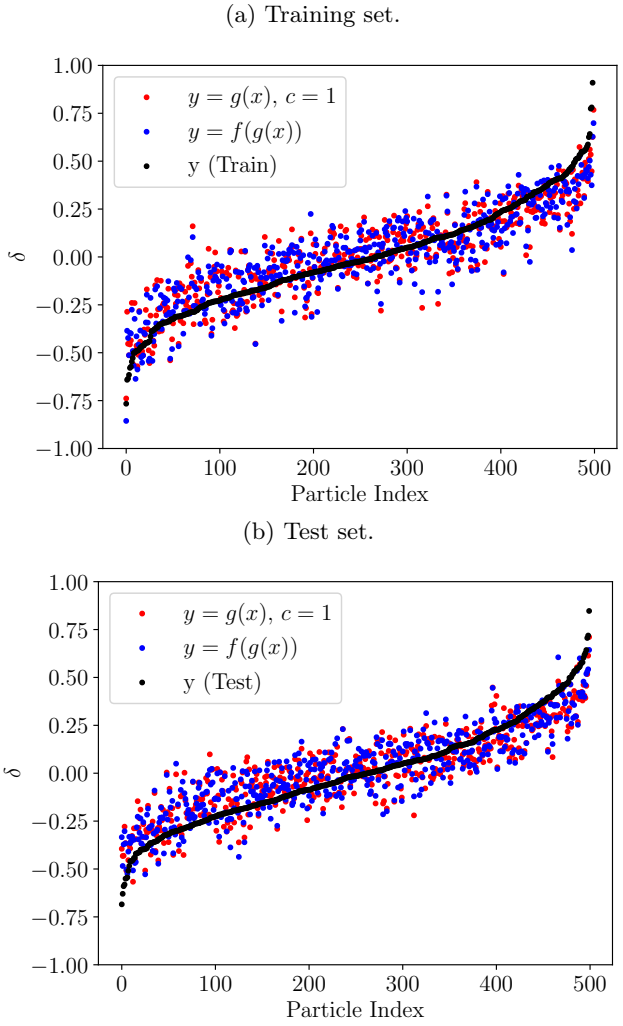


Figure 7.7: Normalized predictions of the deviation from the mean force $\langle F_{fluid} \rangle$, i.e., $\delta = \frac{F_{fluid} - \langle F_{fluid} \rangle}{\langle F_{fluid} \rangle}$. Particles are sorted in ascending order by their target value [223].

ever, this coevolutionary method is susceptible to local optima, which opens space for alternative methods to approximate the interdependencies within the kinematic chain. Related research suggests that GNNs are a viable method to address the IK problem for robotic manipulators [132, 243]. Building on the results from the previous section, which demonstrated the potential of MPNNs to approximate underlying interactions and decompose the problem into smaller subproblems tractable for GP, we aimed to evaluate this approach for the IK problem as well. The scope of this section is to assess the applicability of MPNNs to the IK problem of arbitrary robotics manipulators.

The following section is partly based on the author's contribution to the publication [203].

7.2.1 Translating Manipulators to Graph Structures

Joints translate to nodes, links translate to edges in the MPNN.

We modeled the kinematic chain with links and joints of a robotic manipulator as a graph $g = (V, E)$, consisting of a set of nodes V and edges E. Here, $V = \{\mathbf{v}_i\}_{i=1:N^v}$ denotes the nodes, which in our method corresponded to the joints of the robotic manipulator. The total number of nodes in the graph is denoted by N^v. The node features, $\mathbf{v}_i$, describe characteristics such as joint angle values, joint types, or angular offsets. In our setting, each joint or node contained information about the target pose of the end effector. The edges, $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:N^e}$, represented the connections between these joints. N^e indicated the total number of edges in the graph. The edge feature vector, $\mathbf{e}_k$, included information such as link length or translational offset. MPNNs allow each node to incorporate local structural information and features from

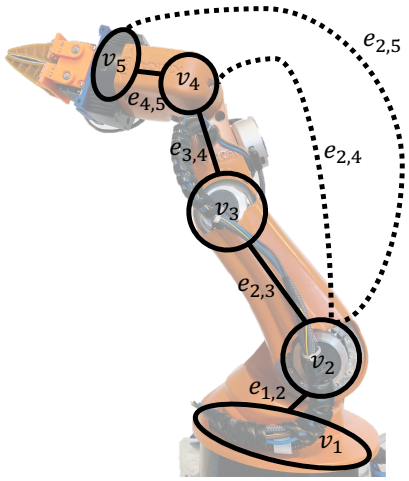


Figure 7.8: Graph projected on a manipulator with 5 DOF. Nodes v_i and v_j are connected by an edge $e_{i,j}$. Solid lines indicate neighborly connections between nodes. Dashed lines indicate the additional edges for full connectivity, using node v_2 as an example to avoid visual clutter. More edges are added for all other nodes in the same way [203].

its surrounding nodes, enabling the network to learn complex dependencies and patterns within the graph. We employed a general MPNN block with an edge model ϕ^e and a node model ϕ^n, which were updated and aggregated according to Eqs. 2.21−2.23 [203].

For many systems, including the one addressed in this section, computations on the edge and node levels suffice, i.e., the update step is complete once Eq. 2.23 is executed. Both ϕ^e and ϕ^n use shared parameters to compute messages and update nodes. In other words, it assumes that the first and second joints interact in the same way as the fourth and fifth.

Different Connectivity Types Within the Graph

The initial application of this approach to arbitrary robotic manipulators proposed multiple variants to experiment with its characteristics and assess its effectiveness [203]. First, the impact of *different underlying connection types* between the joints was evaluated. A *neighborly connection* between joints is inherent to robotic manipulators, where each joint in the kinematic chain is only connected to the previous and subsequent joints through links. This variant relies on local connectivity to learn an IK model, with limited information about the area outside the immediately neighboring joints. Second, a *full connection* variant was assessed, where each joint receives messages from all other joints. Fig. 7.8 shows how the manipulator structure is translated to a graph with neighboring and full connectivity.

Reference-Guided IK

Results reported in [203] indicate that a reference-guided approach delivers better results than a direct estimation of joint angles. It is inspired by robot path planning, where the path from one pose to another is divided into multiple intermediate poses, each close to the previous one. Thus, the learning task shifts, from directly estimating the joint angles towards computing the difference to the previous joint angles to reach the next pose. Mathematically speaking, the predicted vector of angles $\boldsymbol{\theta} = g^{-1}(\mathbf{X}, \boldsymbol{\theta}_{\text{ref}})$ depends on the target pose $\mathbf{X}$ as well as a reference angle $\boldsymbol{\theta}_{\text{ref}}$. It is important to note that we assumed that close intermediate positions also have close joint angles, although this is not necessarily always the case. Since the difference between the neighborly and fully connected graphs were negligible when the reference-guided approach was employed, the focus in this thesis is on a neighboring connection between nodes with reference values for θ_i, which relies on fewer internal interactions between the nodes. The reference angle $\theta_{\text{i,ref}}$ is computed from the true angle θ_i with an added distance randomly sampled from a uniform distribution $\Delta\theta \sim \mathcal{U}(-10°, 10°)$.

7.2.2 Data Generation

Computing a Pose Using the FK and the D-H Convention

To train the MPNN for the IK problem, we generated data using a custom Python-based framework. This framework produces collision-free configura-

Table 7.3: Configuration for 3 DOF manipulators [203].

Joint	θ [°]	θ_{off} [°]	a [cm]	d [cm]	α [°]
1	0-360	0	0	40-60	90
2	0-360	90	24-36	0	0
3	0-360	0	15-20	0	0

Table 7.4: Configuration for 5 DOF manipulators [203].

Joint	θ [°]	θ_{off} [°]	a [cm]	d [cm]	α [°]
1	0-360	0	0	40-60	90
2	0-360	90	24-36	0	0
3	0-360	0	15-20	0	0
4	0-360	-90	9-13	0	-90
5	0-360	0	0	5-8	0

tions for manipulators with user-defined link lengths $\mathbf{U}$ and increasing dataset sizes for 3, 5, and 6 DOF manipulators. The end-effector pose was calculated using the FK equations and the D-H convention. For new instances, a vector $\boldsymbol{\theta}_{new}$ was created, with DOF random values drawn from a uniform distribution within the movement range of each joint [203].

Workspace Coverage

To ensure even coverage of the workspace, we implemented a mechanism that guarantees that new samples are distinct from existing ones in the dataset. A new angle configuration $\boldsymbol{\theta}_{\text{new}}$ was accepted only if Eq. 7.1 was fulfilled, i.e., differed from all existing configurations in the dataset by at least 1° in at least one joint angle:

$$\min_{\overline{\boldsymbol{\theta}} \in \text{Dataset}} \|\overline{\boldsymbol{\theta}} - \boldsymbol{\theta}_{\text{new}}\|_\infty > 1^\circ \tag{7.1}$$

Otherwise, the configuration was rejected. Additionally, configurations that resulted in self-collisions or ground collisions were excluded. This process was repeated until the dataset reached the user-specified size.

Configurations of Link Lengths

The first link U_1 took on values between $40 - 60$cm, and the subsequent link lengths were randomly sampled from the following interval

$$U_{i+1} = [0.75U_i \pm 0.15U_i] \tag{7.2}$$

Thereby, we ensured that the link length U_{i+1} was smaller than U_i, as well as a minimum link length of 5 cm. Tabs. 7.3 and 7.4 show the ranges for all the D-H parameters for 3 and 5 DOF manipulators, respectively. The α value was fixed for all the joints, and the θ value was randomly generated using a uniform distribution between [0,360). The offset values θ_{off} were chosen so that the manipulators took on a specific neutral position when all joint values were equal to 0. The final dataset contained the set of features $\{a_{\text{i}}, d_{\text{i}}, \alpha_{\text{i}}, \theta_{\text{i}}, \theta_{\text{i,off}}, A_{\text{i}}, x, y, z, \Phi, \Theta, \Psi\}$. A_i represented the D-H matrix for each joint, and i denoted the joint index. θ_{i} was available both in degrees and radians [203].

Dataset Types and Sizes

We employed the ten different link length configurations for each DOF using the framework proposed in [203]. The datasets contained 0.5, 1.0, and 2.0 million data points for 3, 5, and 6 DOF manipulators for each configuration, summing up to 5, 10 and 20 million data points, respectively. Additionally, we created holdout test datasets with 10,000 samples for every DOF using link length combinations unused for training, that were within the reachable area of the training datasets, i.e., no extrapolation was required by the MPNN.

7.2.3 Experiment Design

Manipulator Types Assessed in the Experiments

We designed our experiments to assess the performance of the proposed MPNN approach on manipulators with 3 and 5 DOF. We selected 3 DOF as a proof of concept, since this was a comparably simple configuration with known analytical solutions. The 5 DOF manipulator is an unusual configuration, for which we set a comparative baseline in Chapter 4. In the current chapter, we applied

Table 7.5: Parameter Settings for GNNs similar to [203], adapted to our experiments.

Parameter	Settings
Nonlinearity	Rectified Linear Unit (ReLU)
No. hidden layers l	2 (3 DOF), 4 (5 DOF)
No. neurons per layer n	35
Max. learning rate	0.002
Batch size	5000
Optimizer	AdamW
Loss function	Mean Squared Error (MSE)
Training duration	1000 epochs
Early Stopping	10 epochs
Train/validation ratio	0.8 / 0.2, random split
Size of message vector	6
Size of node vector	1 (target variable θ_i)
No. run	31
Node features	x, y, z, Φ, Θ, Ψ, $\theta_{i,\text{off}}$, $\theta_{i,\text{ref}}$, α_i, l_i

Table 7.6: Overview of domain knowledge integrated as bias into the MPNN algorithm for the IK problem.

Bias Type	Bias
Observational	Reference-guided approach with a reference angle θ_{ref} in the training data
Inductive	MPNN as inductive bias to approximate pairwise interactions between joints and subdivide the problem

the proposed algorithm to a wider problem of the same manipulator type, i.e., same DOF and joint types, but varying link lengths. Furthermore, based on the results of [203], we employed a reference-guided approach, which included a reference angle near the target joint angle as inspired by path planning applications.

Parameter Settings

The training data was split in an 80%-20% ratio for training and validation for early stopping detection, which stops the training once the loss on the early stopping set increases over a few defined epochs. We kept the network parameters proposed in [203] and trained the MPNN for 3 DOF with two hidden layers with 22 neurons in each layer. The network for 5 DOF contained 35 neurons per hidden layer and a deeper network with 4 layers to enable the learning of more complex features for the more complex problem. The batch size for training was set to 5000. The weights in the MPNN were initialized randomly. To be able to analyze the sensitivity of the network against initialization, we repeated the training 31 times. Tab. 7.5 summarizes the algorithm configurations for our experiments, and Tab. 7.6 gives an overview of the types of domain knowledge integrated in the proposed algorithm.

7.2.4 Results and Analysis

Central Angle Computation

A crucial part of the analysis was to compute the error between the target and predicted angles, both in terms of the joint angles θ_i and in terms of the resulting orientation. Since the angles ranged between 0° and 360° and lay on a circle, computing the distance using a simple subtraction of values could have led to large errors for small deviations due to the circular nature of angles. For example, although the two angles of 359° and 1° were only 2° apart on the circle, a straightforward subtraction would have yield 358°. To tackle this issue, we employed the so-called *central angles* method, which always returns the absolute value of the minimum distance between two angles. Fig. 7.9 illustrates this method.

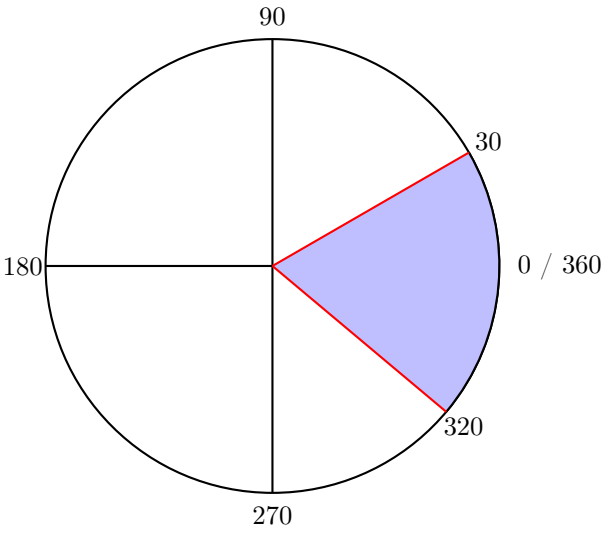


Figure 7.9: The central angle represents the smallest angle difference, such as here when the target angle is 30° and the predicted angle is 320°. The prediction error is thus not 290°, but 70°, as spanned by the blue area.

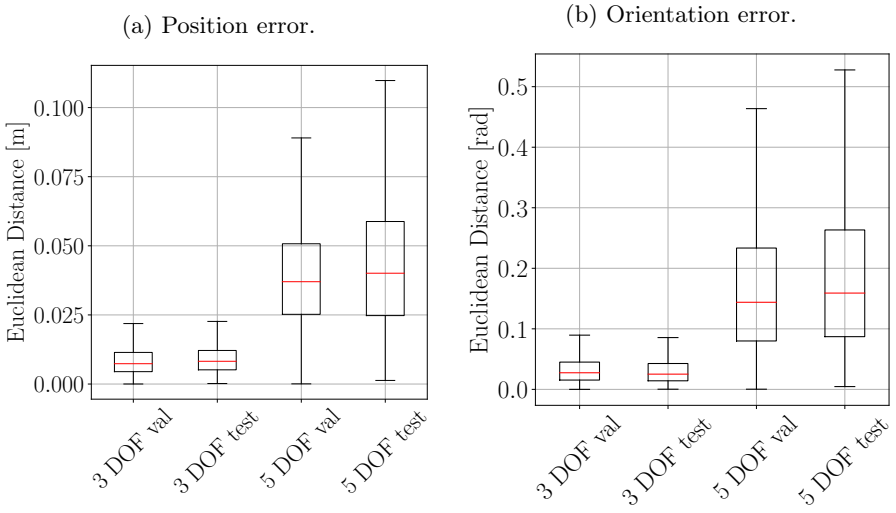


Figure 7.10: Distributions of position and orientation errors in the MPNN prediction for 3 and 5 DOF manipulators on the validation and test datasets.

Prediction Accuracy

Model Selection

From the 31 training runs performed, we observed similar values in the final MSE on the prediction of the joint angles, with mean R^2 values over all joints of 0.99 and higher. We selected the model with the lowest overall prediction MSE of the joint angles based on the validation dataset, incorporating 20% of the actual dataset. While the models were trained on predicting the joint angles correctly given a certain target pose, we analyzed the error in the final pose using the FK equations and the predicted θ values. We moreover assessed the models on the test datasets, which included 10,000 samples of a manipulator with combinations of link lengths not used during training but following a similar pattern, where link lengths decreased from the robot base to the end effector. The total sum of all link lengths remained within the reachable area present in the training data, which ensured that no extrapolation was required. Fig. 7.10 illustrates the position error as the Euclidean distance in three dimensions between the target and predicted poses. It also shows the orientation error, calculated as the Euclidean norm of the central angles for each of the three orientation components.

The position errors were within the expected range of about 1cm *for 3 DOF and* 4cm *for 5 DOF.*

The model for 3 DOF achieved a mean position error of 0.00873 ± 0.00599m, and a mean orientation error of 0.08223 ± 0.46688rad on the validation set. Similar values were reported on the test set, with a mean position error of 0.00923 ± 0.00573m, and mean orientation error of 0.07485 ± 0.44113rad. Thus, the mean deviation in the position was slightly below one centimeter. For 5 DOF, we observed a mean position error of 0.03990 ± 0.02069m, and a mean

orientation error of 0.18746 ± 0.21211rad on the validation set. The error on the test set was slightly higher, with a mean position error of 0.04465 ± 0.02658m, and a mean orientation error of 0.20563 ± 0.22139rad. These observations were supported by Fig. 7.10, where the model for the 5 DOF manipulators exhibited larger median values and a greater spread in the distributions of both position and orientation errors compared to the 3 DOF. Comparing the performances between the validation and test sets, we could, however, conclude that the model for 3 DOF generalized to other manipulator structures with previously unseen combinations of link lengths. For 5 DOF, the generally larger deviations from the target pose, as well as the increase of 0.475cm in the position error between validation and test sets, could have impacted the applicability of this approach. The tolerability of these deviations is dependent on the specific application scenario and the evaluation of domain experts. One can hypothesize that the higher problem complexity of the 5 DOF manipulators compared to 3 DOF was the cause for higher errors. This, however, does not explain the larger errors in terms of orientation compared to the initial results presented in [203]. Due to the black-box nature of the MPNN, further investigation is required to assess the underlying causes of these differences and identify potential adaptations to the model that could mitigate them. To gain further insights into the underlying internal functioning of the models, we assessed the importance of the input features for the edge and node models in the following way.

Analysis of Feature Importance

Why should we analyze the feature importance for this problem?

A major goal of learning the IK problem with an MPNN as an inductive bias for GP is to break the problem into smaller subproblems and reduce the numbers of features considered in each equation. While GP has a limited "built-in" feature selection mechanism, i.e., irrelevant features are simply not present in the resulting equations, the inclusion of too many features can still hamper the results. Given the present node and edge models, we derived a total of nine equations for a 3 DOF manipulator and eleven for a 5 DOF manipulator: one for each of the six message elements, and one for each joint. Each message equation received 14 distinct features as input, and each node equation made use of 16 features, of which six were the message elements. Compared to the original feature space, which consisted of 18 features for 3 DOF and 26 features for 5 DOF, i.e., six pose features and four additional features per joint, this represented a reduction in the number of input variables. In the following, we aim at analyzing the importance of the different features within the node and edge models. The primary goal of this analysis was to understand how features are utilized within the learned models and to identify which ones play the most significant roles in prediction. Given the still relatively large feature space compared to the fluid mechanics problem, and the absence of ground truth equations, we studied the importance values to get a first impression of which features should also be of high relevance within the equations that will substitute these networks. While feature elimination was not the primary objective, we assumed that features with consistently negligible importance, as indicated by importance values below a certain threshold, may contribute little to the model performance. In such cases, removing these features from the feature space could simplify the subsequent symbolic regression step without significantly compromising accuracy. A few studies on feature selection for high-dimensional problems in GP were introduced in Sec. 3.2.5, some of which were concerned with feature permutation to compute a feature importance score [50, 69]. This method is not only applicable for GP, but model-agnostic and thus also effective for the GNNs presented here.

Feature Importance Approach

We implemented a feature permutation approach similar to the one presented in [182]. Within a batch of samples, the features were shuffled one after the other, and the model output using the permuted batch of data was computed. Since we also computed feature importance values for each message element, no ground truth output was available to which the permuted output could be

compared. Thus, we computed a distance between the model output using the original features and the output with a single permuted feature for all features. In other words, we measured how sensitive the model output is to changes in the features. For the message elements, the distance between the undisturbed outputs Y and disturbed outputs $\tilde{Y}$ was computed as the mean absolute value of the difference between them within a batch of size b:

$$\Delta y = \frac{1}{b} \sum_{i=1\ldots b} \left(|Y_i - \tilde{Y}_i| \right) \tag{7.3}$$

We employed the mean absolute error to enable an undistorted analysis of the deviations between the outputs. When features did not contribute to the model output and their permutation causes a tiny change in output $\Delta y < 10^{-2}$, we removed them. The output of each node predicted the joint angle θ. As introduced earlier, the distance in angular space was computed using the central angles instead of the subtraction in Eq. 7.3. We moreover identified and permuted groups of features to assess the influence of multiple features at the same time and account for potential correlations between features. The groups were selected based on combinations of features with the lowest importance values for each node. This made a total of seven groups for the node level for 3 DOF, represented by the variable k, and five groups for the node level for 5 DOF, represented by p:

- $k_1 = \{x, y, z, \Theta, \alpha_i, \theta_{\mathrm{i,ref}}, m_1, m_3, m_4\}$
- $k_2 = \{x, y, z, \Theta, \theta_{\mathrm{i,ref}}, m_1, m_3, m_4\}$
- $k_3 = \{x, y, z, \Theta, \theta_{\mathrm{i,ref}}, m_1\}$
- $k_4 = \{x, y, z, \Theta, \theta_{\mathrm{i,ref}}, \}$
- $k_5 = \{x, y, \theta_{\mathrm{i,ref}}, m_3\}$
- $k_6 = \{\alpha_i, \theta_{\mathrm{i,ref}}\}$
- $k_7 = \{x, \theta_{\mathrm{i,ref}}\}$
- $p_1 = \{x, y\}$
- $p_2 = \{x, y, z\}$
- $p_3 = \{x, y, z, \Phi, \Theta\}$
- $p_4 = \{x, y, z, \Phi, \Theta, l\}$
- $p_5 = \{x, y, z, \Phi, \Theta, l, m_1\}$

Feature Importance on the Message Level for 3 DOF

The feature importance values for the 3 DOF manipulators on the message level are displayed in Fig. 7.11. The plots provide insights into the otherwise hidden network blocks of the MPNN. Since messages are sent from a sender to a receiver node in the MPNN, we disaggregated the values for node features, where the sender was annotated with an s and the receiver with an r subscript. Initially, we could observe a similar importance distribution across all elements, with varying peak values in the most important features. The most important values were typically the three orientation parameters Φ, Θ and Ψ, together with the offset angle of the receiving joint $\theta_{\mathrm{i,off,r}}$, as well as the link length of the sending joint $l_{i,s}$. The output was affected the least when y, $\alpha_{i,r}$, $l_{i,r}$ and $\theta_{\mathrm{i,off,s}}$ were permuted independently of each other. Within the position parameters, x and z showed higher importance than y throughout all message elements. This seemed reasonable given the structure of the 3 DOF manipulator, where the x and y coordinates were coupled, and x implicitly carried the information about y. Within the features of the receiving joint, the offset angle and reference angle were most important across all message elements. Permutation of the link length of the sending joint, moreover, had the highest influence on the model

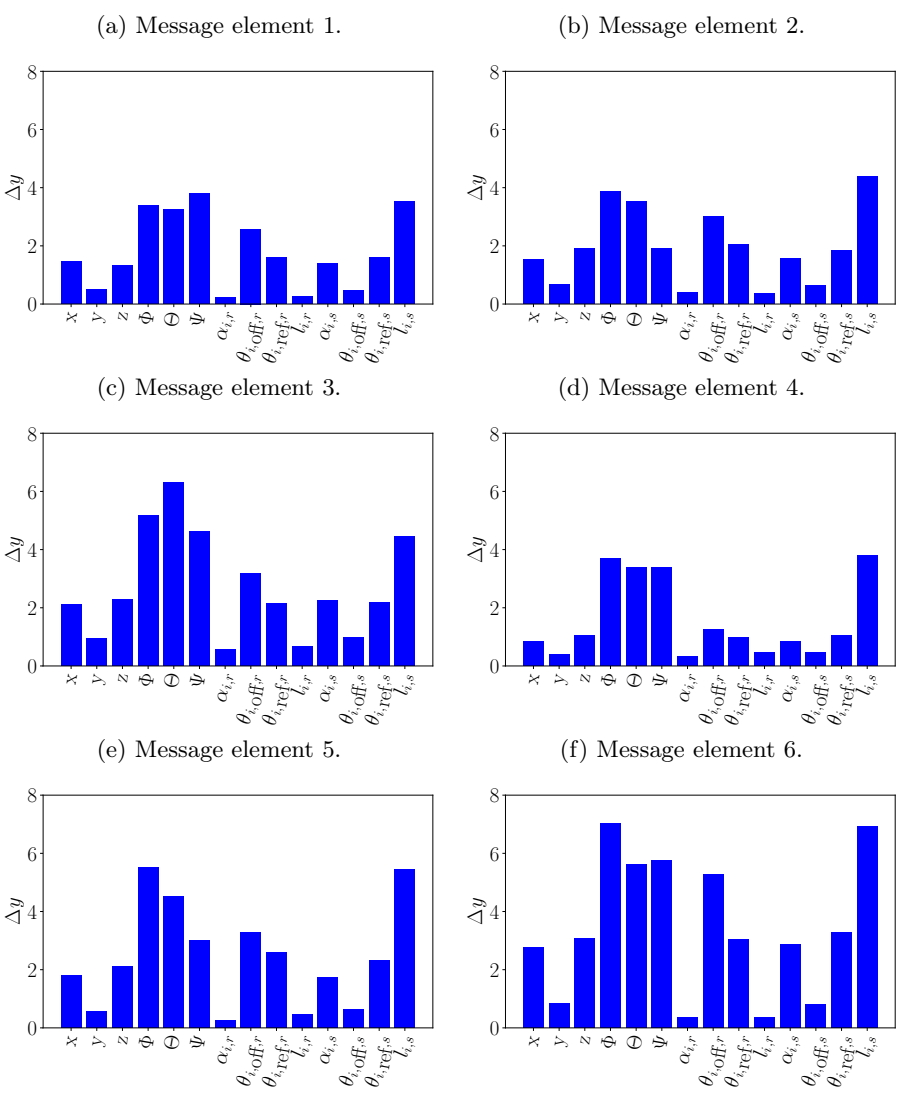


Figure 7.11: Feature importance measurements for message elements of 3 DOF manipulators.

output among the sending joint features. Overall, all the message elements showed a similar distribution of feature importance values, i.e., they reacted similarly when a feature is permuted. While these plots give a rough idea of feature importance, it is important to note that the permutation approach depends on the distribution of the original features. In our dataset, certain features, such as the link length and offset angles, did not follow a continuous distribution, but were instead restricted to a discrete set of specific values. For example, the offset angles only took on values of 0° or 90°. As a result, shuffling these features during the permutation process would have inevitably led to significant changes in their values, which in return could have led to large deviations in the output, which, in turn, could have resulted in large deviations in the output if the feature had indeed been important. Therefore, while the link length and offset angles appeared to be important, their importance should be interpreted with caution, as the exact extent remained unclear.

Feature Importance on the Message Level for 5 DOF

The feature importance values for the 5 DOF manipulators on the message level are displayed in Fig. 7.12. Similar to the previous analysis, we can observe that the features Φ, Ψ, $\theta_{\mathrm{i,off,r}}$ and $l_{i,s}$ were of high relevance across all six message elements. Some message elements also showed a certain dependence on the receiver link length $l_{i,r}$. For both manipulator types, the results suggest that the message elements may have captured redundant information, and the model structure could have potentially been simplified without significant loss of performance. This, however, remains the subject of future research. Moreover, all features showed a significant contribution above 10^{-2} to the model

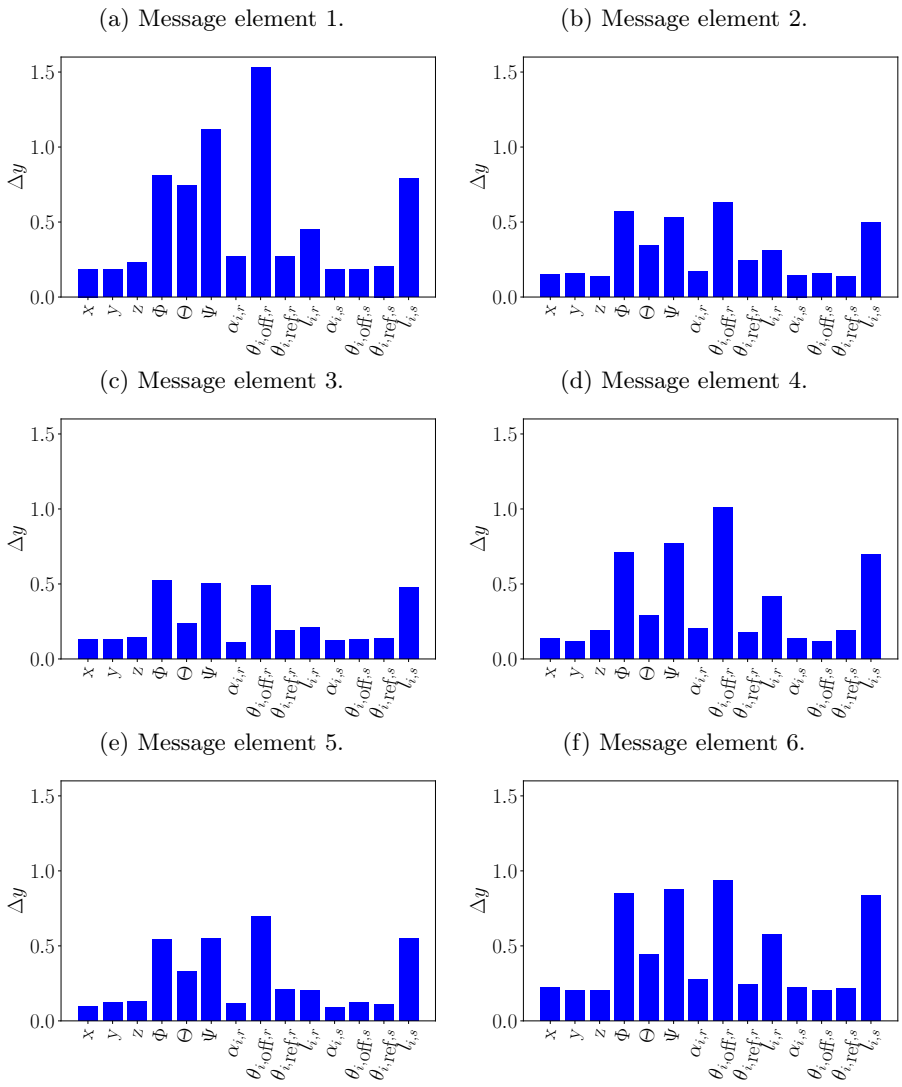


Figure 7.12: Feature importance measurements for message elements of 5 DOF manipulators.

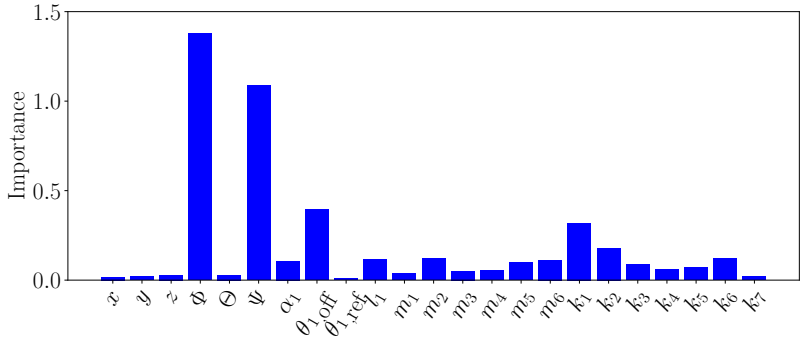
output for both manipulators, so that no feature had to be removed for the subsequent SR.

Feature Importance on the Node Level for 3 DOF

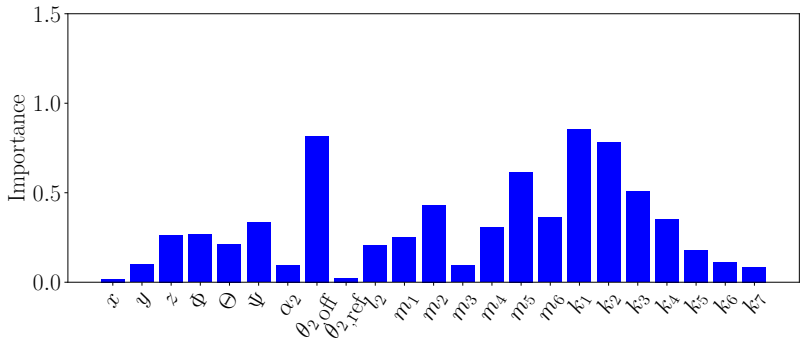
The feature importance on the node level showed higher variations in the distributions for the three joints, as displayed in Fig. 7.13. The predictions of the first joint θ_1 showed the highest sensitivity in their outputs when Φ and Ψ were permuted. Relatively low importance was attributed to the position features x, y, and z, as well as Θ, and the reference joint angle $\theta_{\mathrm{i,\,ref}}$. Moreover, moderate importance was attributed to the offset angle in the first joint, $\theta_{\mathrm{i,off}}$. The message elements m_1, m_3, and m_4 had a lower importance than m_2, m_5 and m_6. However, their contribution to the overall output was comparably small. The groups k_1 to k_4 resembled various combinations of the least important features contributing to θ_1. When the five least important features $k_4 = \{x, y, z, \Theta, \theta_{\mathrm{i,ref}}, \}$ were permuted as a group, the variation in the model output was notably low, and increased when more features were added, as in k_1 - k_3. The high importance values for Φ and Ψ aligned with the unique structure of the 3 DOF manipulator: The orientation component Φ was determined solely in the first joint, as it defined a fixed plane in which the subsequent joints operated. The high importance of Ψ can be attributed to its role in determining whether a position was reached from an overhead configuration. Thus, it defined whether the base joint was oriented towards the target position or the opposite direction. The second and third joints showed a more uniform distribution of importance values over the features. While they were more sensitive to grouped permutations, both showed low importance values when x and the

Figure 7.13: Feature importance measurements for each node/joint in a 3 DOF manipulator.

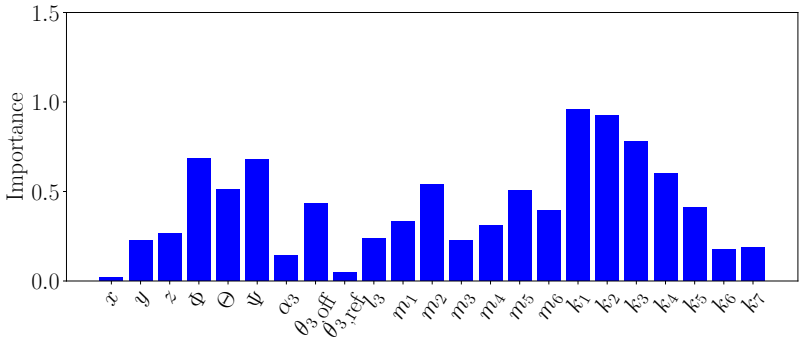
(a) θ_1 feature importance.



(b) θ_2 feature importance.



(c) θ_3 feature importance.



reference joint angle $\theta_{\text{i,ref}}$ were permuted together in group k_7. Overall, it is interesting that the reference joint angles had only little importance on the node level, but a moderate importance on the message level. This suggests that the message elements already included transformations and combinations of these reference angles. Notably, m_2 and m_5 were of greater importance for predicting θ_2 and θ_3 compared to θ_1. However, this analysis did not provide a definitive answer regarding the extent to which the model explicitly utilizes the reference angles to predict the target joint angles.

Feature Importance on the Node Level for 5 DOF

Looking at the feature importance on the node level for manipulators with 5 DOF in Fig. 7.14, one can notice an increased importance of the reference joint angle $\theta_{\text{i,ref}}$ for all joints. Permutation of the three position components x, y, and z, as well as grouped in group p_2, had a negligible effect on the model output, except for θ_2, where z was attributed a higher importance. The higher importance of z in θ_2 can be explained by the fact that θ_2 and θ_3 together built the elbow of the manipulator and required collaboration to achieve a certain angle. We assumed that θ_3 received information about θ_2 through message passing, and thus the two joints cooperated to reach the target z coordinate. The three orientation components Φ, Θ and Ψ were mainly important in the joints θ_1 and θ_5. Within the kinematic chain of the 5 DOF manipulator, these two joints played an important role in determining the final orientation, with

θ_1 fixing the plane in which all subsequent joints operated, and θ_5 allowing for rotation around the yaw axis of the end effector. Overall, the permutation effects of the group of features p_1 were below the threshold 10^{-2} for θ_1 and θ_2, and of group p_2 for the other three joints. Thus, we assumed that the removal of these features would not have a significant impact on the final model output, while simultaneously reducing the search space.

Comparison of 3 DOF and 5 DOF Models

From the analysis in this section, it became apparent that the two MPNN-based models for 3 DOF and 5 DOF differed in the way particular features were used. While the 3 DOF model almost ignored the reference joint angle on the node level, and played a subordinate role on the message level, the model for 5 DOF almost exclusively relied on this reference angle. This observation is in accordance with the results that have been previously published in [203], where the direct estimation of joint angles without a reference angle achieved high accuracy for 3 DOF, but failed to learn a comprehensive model for 5 DOF. Thus, the 3 DOF model is not necessarily reliant on a reference angle, whereas the 5 DOF model requires it to be present in the training features. Given these results, we can infer that the two models operate in very different ways: the 3 DOF model appears to have learned underlying interactions that closely resemble the kinematics of the physical model, while the 5 DOF model seems to rely primarily on the reference angle and learns a compensation function to achieve the target joint angle. One characteristic that both models share is that the feature importance varies for different joints, even though the output is computed by the same shared node network block. This can be explained with the use of ReLU as an activation function, which can turn off neurons in some cases where they are irrelevant.

Consideration on Feature Removal

In this analysis, we set a threshold for an importance of 10^{-2}, below which we assumed that features could be safely omitted without significantly compromising the model accuracy. This value was selected by the domain experts, and depends greatly on the magnitude of the message values as well as the specific characteristics of the problem. The feature importance values presented in this section indicate that there might be further potential in removing certain features from the set of prediction variables. Methods for feature selection are an ongoing topic of research, and determining the most reliable approach to remove features without significantly compromising model performance is highly dependent on the specific problem and model. To make informed decisions regarding the removal of additional features in the future, we propose employing methods such as Shapley values, which quantify the marginal contribution of individual features in a computationally more expensive way.

7.3 Discussion and Limitations

Fluid Mechanics Benchmark

For particle-laden flows, the results demonstrate that the MPNN achieved competitive accuracy when predicting deviations from the mean drag force for varying volume fractions within the Stokes regime, compared to state-of-the-art methods. The equations derived from the edge model outputs were concise, interpretable, and physically meaningful, although they exhibited slightly larger error values than the direct MPNN predictions. Frequent building blocks that appeared in all equations were identified, and made sense from a physics perspective. They did not overfit the training data and offer a viable alternative to ANN-based models, which typically lack interpretability. However, a few equations showed inconsistencies in the dimensional analysis, which will be addressed in the next chapter of this thesis. While our experiments confirmed the applicability of MPNNs as inductive bias for GP regarding the problem of particle-laden flows, the presented models are currently limited to predicting the target variable for a specific volume fraction ϕ. A universal equation that accounts for and is valid across varying volume fractions would be desirable for practical applications, and could potentially contribute to a more profound

understanding of particle-laden flows. Nevertheless, the building blocks identified in Sec. 7.1.6 build a strong foundation for the extension of this approach to higher Reynolds numbers, as these factors could improve equation recovery even for regimes with more turbulent interactions.

Robotics Benchmark

For the IK problem, our focus was on two aspects: evaluating the performance of the MPNN on manipulators with 3 and 5 DOF, and gaining insights into the internal functioning of the network blocks. The position errors were satisfactory and comparable to the ones reported in [203], with mean errors of 1cm for the 3 DOF manipulators and 4cm for the 5 DOF manipulators. The deviations from the target orientation were slightly higher than in the initial report [203]. In the case of the 3 DOF manipulator, the model utilized all available features to varying degrees. Conversely, the 5 DOF model relied heavily on the reference angles, suggesting that the network predominantly learned a compensation function to approximate the difference between the reference and the target angles. This compensation function thus learned the deviation from the reference angle, which in our case was uniformly distributed within $\pm 10°$ from the target angle. However, in scenarios involving robot path planning, where joint configurations may differ by more than $10°$ between consecutive poses, this method might reach its limits. Interestingly, the 3 DOF model assigned comparably low importance to the reference angles, which indicates that the models differed in the underlying approximation of the manipulator kinematics. Both robotic manipulators possessed an inherent limitation: at least one degree of freedom that could not be fulfilled, so that a theoretically infinite number of poses in the workspace could not be reached. This introduced additional complexity to the subsequent symbolic regression task, as the resulting equations had to account for the inherent constraints in the pose space, which are typically highly nonlinear and complex, and therefore difficult to approximate and verify. In our algorithms, these constraints were implicitly included in the dataset as observational bias.

7.4 Summary

In this chapter, we studied MPNNs an as inductive bias for GP, which is a method to tackle high-dimensional problems with interacting entities that can be modeled as graphs. We assessed the potential of this method using the two previously introduced problems: predicting the fluid force acting on particles in particle-laden flows based on simulation data, and approximating the inverse kinematics of non-standard manipulators. In this framework, the MPNN served as a surrogate model, capturing the underlying interactions within the system. Symbolic models were then derived from the MPNN output using a GP algorithm. This introduced a well-motivated bias for certain problems to the underlying model, based on pairwise interactions between entities.

For the problem of particle-laden flows, we included additional domain knowledge in the GP algorithm by employing a complexity measure and imposing constraints on the equation generation process. Furthermore, we preprocessed the data to make them manageable for the GP algorithm. Since the shape of the final model was unknown, we examined two underlying structures: $y = g(x)$ and $y = f(g(x))$. Compared to state-of-the-art approaches, the presented MPNN achieved similar accuracies [16, 255]. The symbolic models consistently performed slightly worse than the MPNN, although errors of both approaches were of the same order of magnitude. A test on unseen data indicated that our models did not overfit. The underlying structure, $y = f(g(x))$, performed best on the provided benchmarks. We moreover identified building blocks which frequently appeared in all equations. The equations also revealed which features were most influential on the target variable. Altogether, our approach offers a promising, human-interpretable alternative to the hidden transformation in ANN blocks. Building upon the results presented in Chapter 5, we scaled up

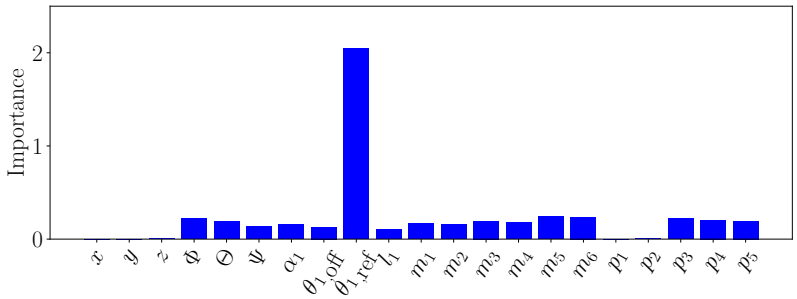
from two to thirty particles.

To approximate the inverse kinematics of non-standard manipulators, we trained MPNNs for 3 and 5 DOF manipulators to predict the joint angles given a specific target pose. We employed a reference-guided approach, which included a reference angle θ_{ref} close to the target angle θ, which demonstrated accurate results in [203]. The resulting position errors were within the expected range of about 1cm for 3 DOF and 4cm for 5 DOF. The MPNN showed higher mean deviations from the target orientation, with 0.082rad for 3 DOF and 0.187rad for 5 DOF. Generally, the 3 DOF manipulator seemed to be easier to learn compared to the 5 DOF manipulator with two more joints in the kinematic chain. Differences in the underlying models became apparent when the feature importance on the message and node levels were analyzed using an approach based on feature permutation. While the 3 DOF model employed all features and message elements to a certain extent, the predictions of the node model for 5 DOF relied heavily on the reference joint angle for all joints. This opened space for the removal of features with low importance for the 5 DOF manipulator in the symbolic regression step.

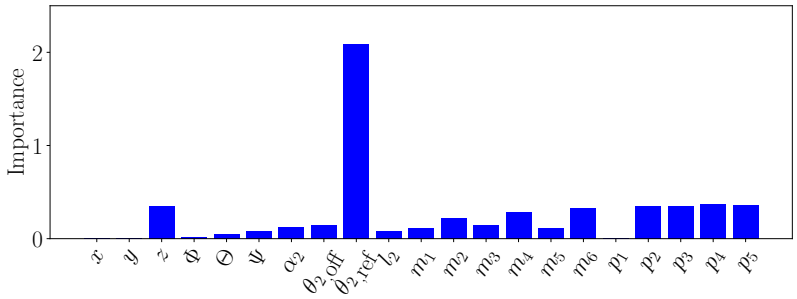
The results presented in this chapter overall confirm the applicability of MPNNs to the problems addressed in this thesis, though further verification is required for the 5-DOF manipulator. In the following chapter, we will focus on the symbolic regression part of the learning pipeline and develop unit-conformal equations for both problems.

Figure 7.14: Feature importance measurements for each node/joint in a 5 DOF manipulator. The variable p refers to groups of features.

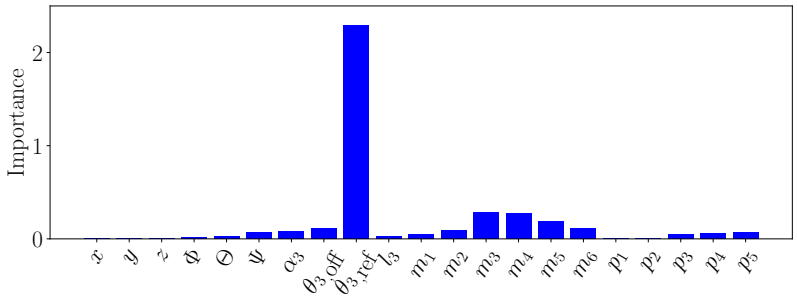
(a) θ_1 feature importance.



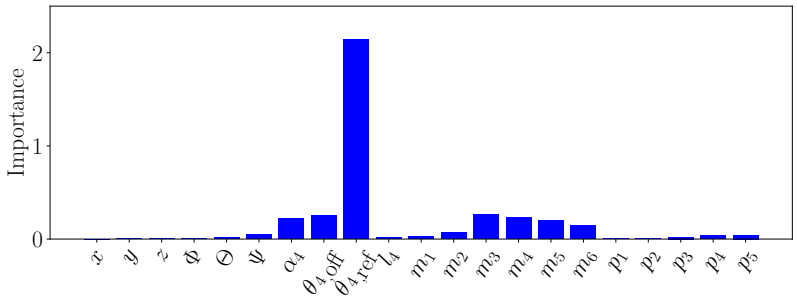
(b) θ_2 feature importance.



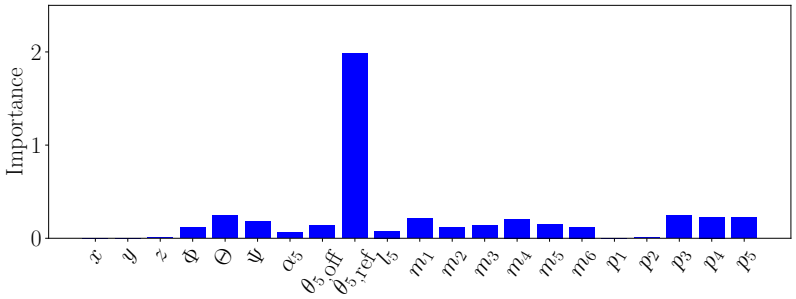
(c) θ_3 feature importance.



(d) θ_4 feature importance.



(e) θ_5 feature importance.



8 Unit-Aware Genetic Programming for Symbolic Regression

Introduction and Chapter Goals

In this chapter, we introduce novel methods for unit-aware GP to address limitations in existing studies and frameworks. As discussed in Sec. 3.3.2, there is extensive literature on unit-aware SR available where units of constants are known a priori or generally treated as dimensionless. However, a largely overlooked issue is incorporating new constants, which are learned during evolution, into dimensional analysis. The unknown units of these constants make it challenging to apply traditional dimensional analysis and constraint-handling techniques. While most equations presented in Sec. 7.1.6 did not exhibit unit violations, the method applied relied on expert-defined constraints for the building rules and controlling function nesting. This may not always be available in advance or could restrict the search space in an undesirable way, which further motivated the investigation of algorithmic techniques for unit-aware GP. A recently published algorithm by Cranmer et al. [57], considered unknown constants as "wildcards" in the dimensional analysis, and formed the foundation of the methods presented in this chapter. Our goal for this chapter is to assess different constraint handling methods for unit-aware GP with new constants and units.

8.1 Unit-Aware Genetic Programming with Unknown Constants

The following section is largely based on the author's publication [224].

Our contribution was inspired by early and recent developments in unit-aware GP.

One of the earliest contributions to unit-aware GP was made by Keijzer et al. [128], in which they evaluated various constraint-handling methods to address violations of physical laws in symbolic models. Unlike their approach, however, our method did not assume that new constants are dimensionless, which significantly impacted the dimensional analysis. We built upon this work and assessed different methods to handle unit violations using a dimensional analysis function that accounted for unknown constants. We used the unit propagation scheme by Cranmer et al. [57] as a starting point for our dimensional analysis procedure. In their approach, a large penalty value was assigned to the primary objective of the algorithm, similar to a death penalty. However, it still used computational resources for the expensive parameter estimation [224].

Dimensional analysis is cheaper than parameter fitting.

Our proposed constraint handling approaches exploited the cheaper dimensional analysis to handle unit violations before fitting. We considered the

Table 8.1: Set of common operation used in GP with their expected input units and the resulting output unit. A joker unit is represented as $[\diamondsuit, \diamondsuit, \diamondsuit]$ (similar to [224], with arc functions added).

Operation	Units of operands	Output unit of operation
$+, -$	$[a, b, c]$, $[a, b, c]$	$[a, b, c]$
	$[a, b, c]$, $[\diamondsuit, \diamondsuit, \diamondsuit]$	$[a, b, c]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$, $[\diamondsuit, \diamondsuit, \diamondsuit]$	$[\diamondsuit, \diamondsuit, \diamondsuit]$
$\cdot$	$[a, b, c]$, $[d, e, f]$	$[a + d, b + e, c + f]$
	$[a, b, c]$, $[\diamondsuit, \diamondsuit, \diamondsuit]$	$[\diamondsuit, \diamondsuit, \diamondsuit]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$, $[\diamondsuit, \diamondsuit, \diamondsuit]$	$[\diamondsuit, \diamondsuit, \diamondsuit]$
$\div$	$[a, b, c]$, $[d, e, f]$	$[a - d, b - e, c - f]$
	$[a, b, c]$, $[\diamondsuit, \diamondsuit, \diamondsuit]$	$[\diamondsuit, \diamondsuit, \diamondsuit]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$, $[\diamondsuit, \diamondsuit, \diamondsuit]$	$[\diamondsuit, \diamondsuit, \diamondsuit]$
e°, $\log(\circ)$	$[0, 0, 0]$	$[0, 0, 0]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$	$[0,\ 0,\ 0]$
$\sin(\circ)$, $\cos(\circ)$, $\tan(\circ)$	$[0, 0, 0]$	$[0, 0, 0]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$	$[0, 0, 0]$
$\arcsin(\circ)$, $\arccos(\circ)$	$[0, 0, 0]$	$[0, 0, 0]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$	$[0, 0, 0]$
$\arctan 2(\frac{\circ}{\circ})$	$[a, b, c]$, $[a, b, c]$	$[0, 0, 0]$
	$[a, b, c]$, $[\diamondsuit, \diamondsuit, \diamondsuit]$	$[0, 0, 0]$
$\sqrt{\circ}$	$[a, b, c]$	$[\frac{a}{2}, \frac{b}{2}, \frac{c}{2}]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$	$[\diamondsuit, \diamondsuit, \diamondsuit]$
$\circ^k$ (k $\in \mathbf{N}$)	$[a, b, c]$	$[a \cdot k, b \cdot k, c \cdot k]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$	$[\diamondsuit, \diamondsuit, \diamondsuit]$
$\circ^\circ$ (binary power)	$[0, 0, 0]$, $[0, 0, 0]$	$[0, 0, 0]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$, $[0, 0, 0]$	$[0, 0, 0]$
	$[\diamondsuit, \diamondsuit, \diamondsuit]$, $[\diamondsuit, \diamondsuit, \diamondsuit]$	$[0, 0, 0]$

number of unit violations in the dimensional analysis, rather than returning a boolean value that indicates whether a violation occurs. We proposed and compared different constraint handling methods to eliminate unit violations in the evolutionary process. We furthermore assessed a multi-objective approach, minimizing the magnitude of unit violations rather than completely removing them [224].

We studied different constraint handling methods using datasets with known and unknown ground truth and varying noise levels.

We studied the effect of these techniques using datasets of equations that had been discovered empirically in the past by influential scientists like Newton and Kepler. We also applied different noise levels to the benchmark datasets to examine how sensitive our approaches are to noisy data. We know from related studies that the importance of prior knowledge increased as the noise in the data increases [104, 128]. We aimed to study whether this effect is also observable when constants with unknown units are used [224]. We furthermore tested the proposed methods on datasets without ground truth from thermodynamics, as well as an extensive study on the final step of the proposed learning pipeline for particle-laden flows and inverse kinematics.

8.1.1 Dimensional Analysis with Unknown Constants

Constants in GP

In GP, different types of constants are used in practice. When only known constants are used, they can be included in the training data and treated like regular features (e.g., the Feynman datasets [272]). When unknown constants are used, contemporary GP-based SR methods use parameter estimation on top of the evolutionary process. The number and position of constants within

an equation is determined during the generation of a tree. The values of the constants are then fitted to the target variable using a parameter estimation algorithm, such as the BFGS algorithm (as in `PySR` [57]) or the Levenberg-Marquardt method (as in `TiSR` [170]). This fitting process is a computationally expensive task. In the following, we take a closer look at the unit propagation scheme, including such new constants.

Unit Vector Representation in Dimensional Analysis

The units of a variable can be expressed as a vector of exponents of SI units with the order [m, kg, s, A, K, mol, cd]. A quantity in Newton [N] = $[\frac{\text{kg}\cdot\text{m}}{\text{s}^2}]$ can thus be expressed as $[1, 1, -2, 0, 0, 0, 0]$. In the subsequently studied scenarios, constants had generally unknown units, which made traditional approaches to detecting unit violations infeasible. To overcome this issue, we applied the unit propagation scheme, similarly as implemented in `SymbolicRegression.jl`, and introduced a joker unit $[\diamondsuit, \diamondsuit, \diamondsuit]$, which represents unknown units [57]. Dimensionless inputs are expressed as $[0, 0, 0]$. Table 8.1 displays how operands with known and unknown units are handled for a set of functions that are commonly used in SR algorithms. This set of functions is non-exhaustive and can be extended to custom functions as well. For the sake of readability, we display only three elements of the unit vector. The rules, however, apply to all seven elements.

Handling Joker Units in Dimensional Analysis

The use of joker units leads to some special cases which have to be addressed: addition and subtraction require equal units of both operands. If one operand is a joker, the unit of the other operand is returned. If both operands are jokers, a joker is returned. For multiplication and division, one or two joker operands produce a joker output. Functions requiring dimensionless inputs assume that a joker operand is dimensionless, and return a dimensionless quantity accordingly. Operations with fixed exponents ($\sqrt{\circ}$ and power operations $\circ^2, \circ^3, \dots$) produce a joker output if the function input is a joker. The binary power operator requires both operands to be dimensionless and returns a dimensionless quantity. If one or two operands have joker units, they are assumed to be dimensionless to return a dimensionless quantity.

Tree Traversal Algorithm for Dimensional Analysis

The recursive Algorithm 7 for dimensional analysis traverses the tree in the most straightforward way, like the evaluation itself, starting at the root node. The joker unit is only introduced into the tree by constants. As Table 8.1 indicates, these jokers are propagated through the tree by most of the functions. Unit violations occur when operands with non-matching or non-joker units are added or subtracted, as well as for functions which require dimensionless inputs. When a violation occurs, the violation counter is increased by one (see lines 15, 28, 33), and the true output unit of the operation is returned. In the case of addition and subtraction, one of the operand units is chosen randomly. For example, the term $\log([1, 2, 0])$ violates the rules defined in Table 8.1. In this case, the true output unit $[0, 0, 0]$ of the operation is returned. Fig. 8.1 displays two exemplary GP trees with and without unit violations according to the proposed dimensional analysis and tree traversal algorithm.

Mismatches Between Output and Target Units

Once the traversal is completed, the algorithm returns the output unit d of the equation as well as the number of unit violations v. The Manhattan distance between d and the target unit d' is added to v to also account for mismatches with the target unit. A joker output is assumed to be equal to the target unit.

8.1.2 Techniques to Handle Unit Violations in Symbolic Models

As derived from the literature review in Sec. 3.3.2, we introduced three techniques to deal with unit violations in GP trees.

Evolutive Culling

Removal of Unit-Violating Individuals Before Constant Fitting.

The dimensional analysis is computationally cheaper compared to the fitting of constants followed by the numerical evaluation. Our evolutive culling approach made use of this fact by performing the dimensional analysis directly after an offspring was created. Individuals with unit violations were excluded from the

Algorithm 7: Recursive Dimensional Analysis (with slight adaptations from [224])

input : Root node n of the tree, units of variables
output : Tuple (output dimension d, number of unit violations v)

```
function recDimAnalysis(n):
    if n is a constant then
        return([◇, ◇, ◇], 0)
    end
    if n is a variable then
        return([a, b, c], 0)
    end
    if n is a unary operation then
        d, v ← recDimAnalysis(n.child)
        if units match case from Table 8.1 then
            d ← unit after execution of operation
        else
            d ← true output unit of operation
            v ← v + 1
        end
        return (d, v)
    end
    if n is a binary operation then
        d_right, v_right ← recDimAnalysis(n.right)
        d_left, v_left ← recDimAnalysis(n.left)
        if units match case from Table 8.1 then
            d ← unit after execution of operation
            v ← v_right + v_left
        else if n ∈ {+, −} then
            v ← v_right + v_left + 1
            d ← choice(d_left, d_right)
        else
            v ← v_right + v_left + 1
            d ← true output unit of operation
        end
        return (d, v)
    end
end
```

population. Compared to the death penalty approach, this method saved time by avoiding fitting and evaluating an invalid model that will not survive the next generation because of the high penalty given to the primary objective. Thus, the space of valid individuals can be explored more thoroughly. As a potential disadvantage, individuals with high accuracy but small unit violations could not evolve into individuals without unit violations. This might lead to overall worse performance regarding the primary objective [224].

Repair Mechanism

Unit Violations as a Hint for Constant Insertion

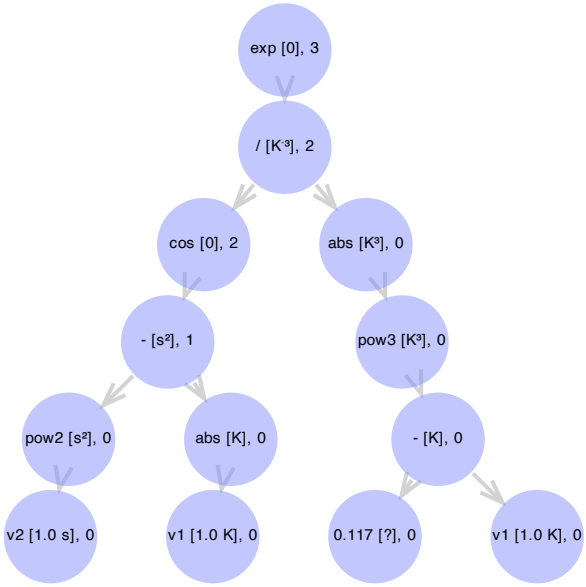
For many fundamental laws of physics, constants alongside their units had to be discovered empirically to fit experimental observations. These multiplicative constants often have unconventional units, which balance output units of an equation to match the target unit. Vice versa, a unit violation can be considered a hint where such a balancing constant should be inserted. We proposed the following repair mechanism: whenever a unit violation occurs, a multiplicative constant is inserted into the tree at that position to match the expected unit of the function [224].

Repair by Insertion of Joker Units Through Constants

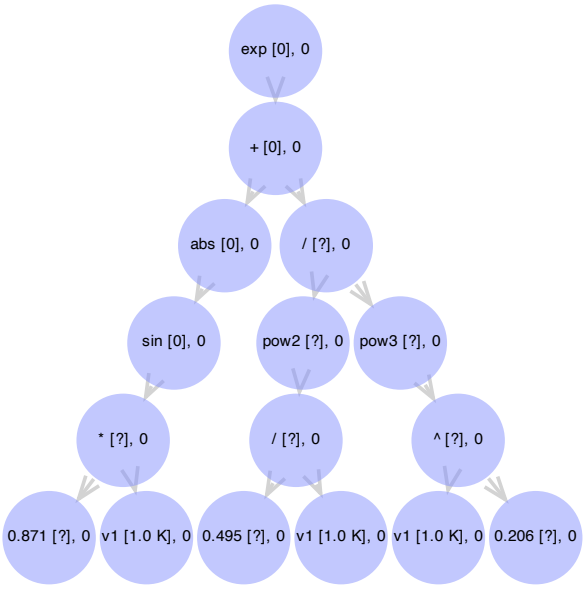
Algorithm 7 was modified so that a multiplicative constant was inserted whenever a unit violation was detected, rather than increasing the violation counter. For example, an addition of [m] and [s] could be balanced by multiplying one of the operands with a constant. This turned the term into a joker so that the

Figure 8.1: Example of a GP tree with four variables v1, v2, v3, v4 and units associated with each variable and operation in angle brackets. Jokes units are displayed as [?], dimensionless units as [0]. The value after the comma is the aggregated dimension error at each node.

(a) GP tree with unit violations according to the unit propagation scheme, aggregated throughout the tree.



(b) GP tree without unit violations according to the unit propagation scheme. Potential unit mismatches can be balanced by joker units in this example.



function returned the unit of the other operand according to Tab. 8.1. The operand that was repaired was chosen randomly, which influenced the number of inserted constants as it was propagated further through the tree. When functions that expected dimensionless input received an incorrect unit, the input term was multiplied by a constant to render it dimensionless. Since the repair function was applied immediately after the offspring generation, only valid individuals were considered. The repaired trees then went into the fitting and evaluation process [224]. As a potential downside, the repair mechanism could have led to the insertion of many or unnecessary constants, which might have negatively affected the primary objective and slowed down the fitting process.

Multi-Objective Approach

Leaving the Space of Valid Models Towards Multi-Objective Constraint Handling

The two methods discussed previously focused on exploring the space of valid, physics-adherent equations. The multi-objective approach presented here allows for unit-violating individuals within the population, and it considers the number of unit violation as an additional objective.

Minimizing the Number of Unit Violations as an Additional Objective

Multi-objective optimization makes use of the concept of Pareto-dominance, as introduced in Sec. 2.3.2. Contemporary GP algorithms minimize multiple objectives at the same time, usually an error and a complexity objective. Depending on the application, it can be beneficial to include a correlation measure as a supporting objective. This helps individuals with poor accuracy but high correlation with the target variable to advance to the next generation, where they can continue evolving to better individuals. Formulating model constraints as additional optimization objectives is a common approach in GP [104, 128, 299]. We employed the NSGA-II algorithm to optimize multiple objectives simultaneously [65]. The resulting *Pareto-optimal* (PO) front contained multiple equations of the same level of complexity — with and without unit violations. Equations without unit violations are generally preferred over equations with unit violations if they have the same accuracy and complexity. This approach encouraged the algorithm to evolve towards physically meaningful equations. However, there was no guarantee that a model without unit violations were found for each level of complexity.

Software Implementation

All algorithms were implemented in `TiSR`, a GP-based framework for thermodynamics-informed symbolic regression [170] written in Julia. Its applicability is not limited to thermodynamics, but any kind of problems from the science and engineering domain. `TiSR` allows for fast algorithmic prototyping through simple code structures, while including all state-of-the-art components of a GP-based SR framework.

8.2 Datasets and Experiment Configurations

8.2.1 Datasets

`empiricalBench` Datasets

The proposed algorithms were evaluated on known empirical equations from the `empiricalBench` benchmark presented in [57]. This physics benchmark does not include constants in the datasets so that the algorithms have to recover them alongside the form of the target equation. Table 8.2 gives an overview of selected datasets for which dimensional analysis can be performed. In addition, we used datasets from physics applications without ground truth [224].

Addition of Varying Noise Values to Output Variable

We also studied the sensitivity of the proposed algorithms to noise. When recovering the exact equation on noisy data, the choice of the noise level is an important parameter. It has to be guaranteed that the noisy data is still described best by the target equation, and not a different one of the same complexity. We assumed that beyond 10% noise, it is difficult to recover the exact equation. The noise levels of 5% and 10% were inspired by publications which also study the capabilities of an algorithm to recover a specific equation [e.g. 91, 152]. For the Rydberg equation, noise levels beyond 3% were too noisy for the exact equation to be recovered, as experiments with 10% noise indicated [57]. We thus applied 1% and 3% noise [224].

Thermodynamics Dataset

We moreover employed a real-world dataset based on measurements from the area of thermodynamics to study the effect of the unit-aware algorithm on problems with unknown ground truth. It used the temperature T and density ρ of a gas mix to predict the pressure P [284].

Fluid Mechanics Dataset

For the fluid mechanics problem, we employed a dataset from the application of particle-laden flows as introduced in [79], which includes varying volume fractions ϕ. The dataset was generated using a similar method to that described in Sec. 7.1.3, but with more realistic particle arrangements. Specifically, the array of particles in the free stream is no longer located in an otherwise empty space. Rather, it has a theoretically infinite number of neighbors. Details on the exact data generation can be found in [79]. The features in the dataset, however, remained the same as in Sec. 7.1.3. The drag force F_D on a particle is computed from the positions of its neighboring particles in spherical coordinates r, θ, φ.

Table 8.2: Benchmark equations employed for our experiments with their input and target features and the respective units (similar to [224], with robotics dataset added).

Name	Equation	Input Features & Units	Target Unit
Hubble's Law	$v = H_0 D$	Distance D [m]	Velocity v [$\mathrm{m\,s^{-1}}$]
Kepler's Third Law	$P = (\circ)\sqrt{a^3}$	Distance a [m]	Period P [d]
Newton's Gravitation	$F = G\frac{m_1 m_2}{r^2}$	Mass m_1, m_2 [kg], Distance r [m]	Force F [N]
Ideal Gas Law	$P = \frac{nRT}{V}$	Number density n [mol], Temperature T [K], Volume V [$\mathrm{m^3}$]	Pressure P [Pa]
Rydberg Formula	$\lambda = \frac{1}{R_H(\frac{1}{n_1^2} - \frac{1}{n_2^2})}$	Principal Quantum Number n_1, n_2 [·]	Wavelength λ [m]
Fluid Mechanics	unknown	Distance r [m], Angle θ, φ [·]	Force F [N]
Thermodynamics	unknown	Temperature T [K], Density ρ [$\mathrm{kg\,m^{-3}}$]	Pressure P [Pa]

Table 8.3: Algorithm configurations for experiments [224].

Parameter	Settings
Population size	500
Max. complexity of equations	30
Complexity of variables and functions	1
Complexity of constants	2
Function set empiricalBench	$+, -, \cdot, \div, e^\circ, \log(\circ), \sqrt{\circ}, \circ^2, \circ^3$
Function set Fluid Mechanics	$+, -, \cdot, \div, e^\circ, \log(\circ), \sin(\circ), \cos(\circ), \circ^\circ$
Function set Thermodynamics	$+, -, \cdot, \div, e^\circ, \log(\circ), \circ^\circ$

8.2.2 Experiment Configurations

Table 8.3 gives an overview of the algorithm settings and use-case dependent function sets. The input features and units from Table 8.2 were the training data of the algorithms, so that necessary constants had to be identified by the algorithms. For other parameters, the standard settings of `TiSR` were used [170]. We set time limits of thirty minutes for experiments on `empiricalBench` datasets and sixty minutes for experiments without ground truth. This approach was preferred over fixed generation counts due to algorithmic modifications that affect the generation runtime. However, we aimed to evolve unit-adherent equations without compromising runtime efficiency. We compared the proposed algorithm to a baseline algorithm without dimensional analysis. All algorithms optimized multiple objectives at the same time: the MSE, the function complexity and the Spearman correlation as a supporting objective as defined in [299]. In addition, we assessed an algorithm variant that minimized the number of unit violations as a fourth objective. Each algorithm was repeated 31 times [224].

Table 8.4: Number of correct/almost correct/wrong rediscoveries of target equations for different datasets and noise levels out of 31 runs [224].

Dataset	Noise	Baseline	Evolutive Culling	Repair Mechanism	Multi-objective
Hubble	0%	31/0/0	31/0/0	30/1/0	31/0/0
	5%	31/0/0	31/0/0	28/3/0	31/0/0
	10%	31/0/0	31/0/0	26/5/0	31/0/0
Kepler	0%	31/0/0	31/0/0	31/0/0	31/0/0
	5%	31/0/0	31/0/0	25/6/0	31/0/0
	10%	31/0/0	30/1/0	26/5/0	31/0/0
Newton	0%	31/0/0	31/0/0	31/0/0	31/0/0
	5%	31/0/0	31/0/0	31/0/0	31/0/0
	10%	31/0/0	31/0/0	31/0/0	31/0/0
Ideal Gas	0%	31/0/0	31/0/0	31/0/0	31/0/0
	5%	30/1/0	31/0/0	16/17/0	31/0/0
	10%	31/0/0	31/0/0	8/23/0	31/0/0
Rydberg	0%	31/0/0	31/0/0	31/0/0	29/0/2
	1%	31/0/0	31/0/0	31/0/0	31/0/0
	3%	27/3/1	29/1/1	24/4/3	29/1/1

8.2.3 Evaluation Procedure

Ensuring Equivalence Between Target and Predicted Equations

The assessment whether an algorithm identified a specific target equation correctly comes with two major issues: first, the selection of a solution from the PO front. Finding the best trade-off between accuracy and complexity automatically is a complex task. And second, the equivalence check of two equations using libraries like Python `sympy` or Julia `SymbolicUtils`. As related studies reported [152], small differences in the simplification as well as the value of fitted constants might lead to misclassification. To overcome these issues and base our analysis on trustworthy results, we eye-checked each PO front for the target equation, which makes a total of more than 1800 checked PO fronts. Some parts of the analysis could be accelerated by automatically scanning a PO front for solutions which have already been classified as correct by a human. An equation counted as solved when the shape of the equation was correct, the exact values of the fitted constants were irrelevant. Since the constant fitting process is a multimodal problem and an optimum cannot be guaranteed, we preferred this method to evaluate the actual quality of the GP algorithm. We defined two stages of success: finding the exact solution, and finding a solution close to the exact one, which was measured by eyeball. For the datasets without ground truth, we analyzed the PO fronts [224]. As they are of major interest in this thesis, the equations developed for the fluid mechanics and robotics datasets will be further examined in Sec. 8.4 and Sec. 8.5 respectively.

8.3 Results and Analysis

8.3.1 Datasets with Known Ground Truth

All algorithms had a high rate of correct equation discovery.

Table 8.4 gives an overview of the performance of the proposed algorithms on known benchmark datasets of equations, which have been derived empirically in the past. It becomes apparent that all algorithms recovered the correct equations for all datasets and all noise levels with a high success rate. Only the proposed repair mechanism had lower rates of identifying the exact equation as the noise level increases. It still found solutions close to the target equation

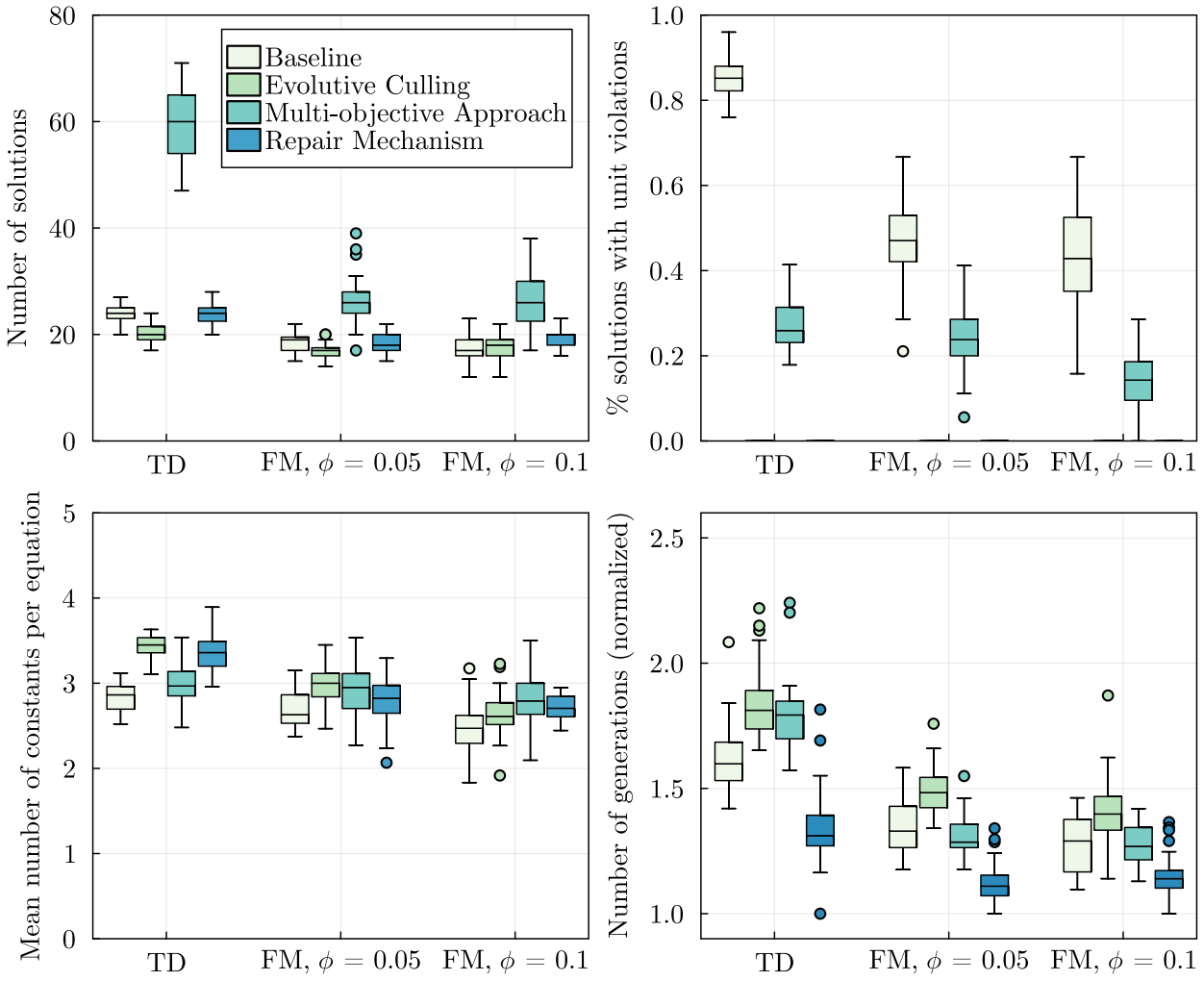


Figure 8.2: Measurements on the Pareto-optimal front for datasets with unknown solutions from thermodynamics and fluid mechanics over 31 independent runs [224].

in the final PO front, which often contained additional constants. Overall, we concluded from these results that evolutive culling as well as the multi-objective approach performed at least as good as the baseline method. However, it should be noted that there was almost no space for improvement, as the baseline algorithm found the correct solution in almost all cases [224].

8.3.2 Datasets with Unknown Ground Truth

Thermodynamics and fluid mechanics results are analyzed in this section.

In this section, we focus on the TD and FM datasets to analyze and compare the performance of the algorithms on problems without ground truth. To compare the algorithms, we analyzed the resulting PO fronts for interesting characteristics: the numbers of solutions, the percentage of solutions with unit violations, and the mean number of constants in the equation. Furthermore, we looked at the number of generations performed within the time limit. For pairwise statistical comparison to the baseline method, the non-parametric Mann-Whitney U test at a confidence level $\alpha = 0.95$ was performed [224].

The multi-objective approach had more solutions in the PO front.

The upper left plot in Fig. 8.2 indicates that the PO fronts of the multi-objective approach contained more solutions compared to the other approaches, which is supported by statistical tests. This can be explained by the additional unit violation objective, which allows the algorithm to include solutions with multiple levels of unit violations per complexity value. On the TD problem, the size of the PO front exceeded 60 solutions in some cases, which increased the challenge for decision makers to select a single solution as the final result of the algorithm [224].

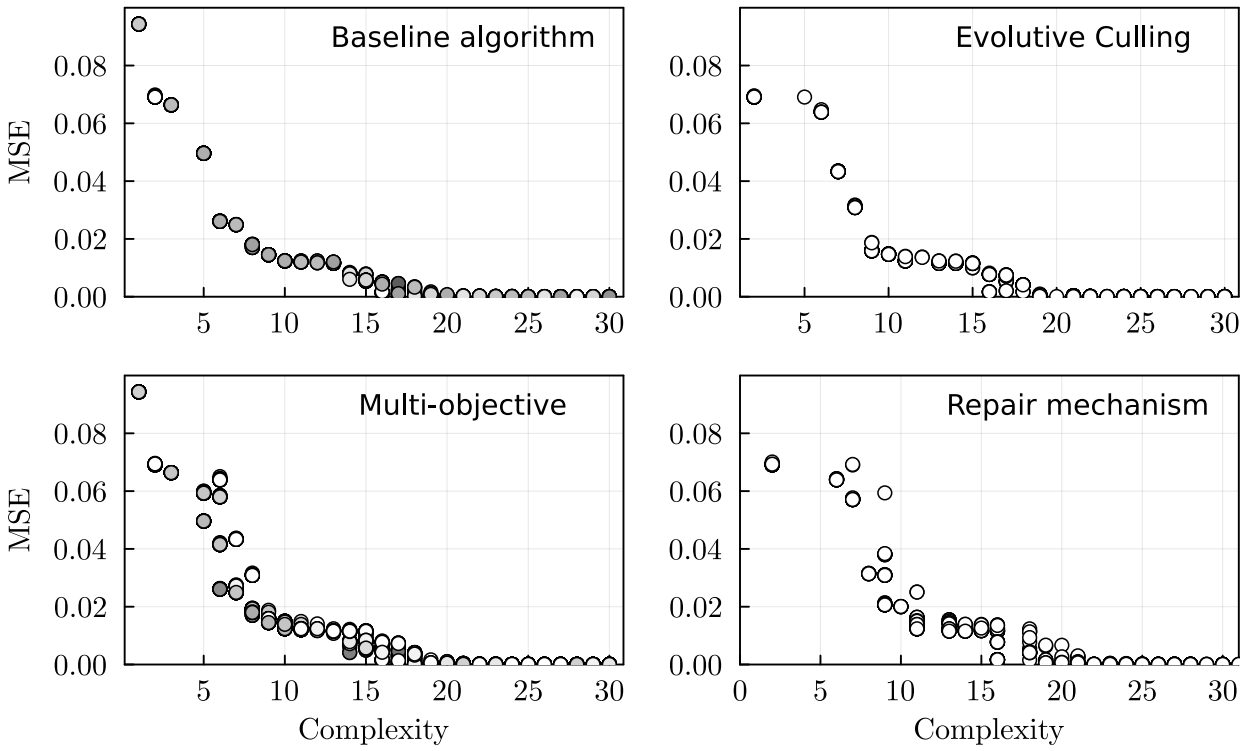


Figure 8.3: Solutions of 31 combined PO fronts per algorithm on the TD dataset. The magnitude of unit violations is color-coded from white (0 violations) to black (22 violations), with 22 being the maximum number of unit violations on the TD dataset [224].

The upper right subplot of Fig. 8.2 shows the percentage of solutions with unit violations within the PO front. When using the multi-objective algorithm, if a complexity level had multiple solutions with varying numbers of unit violations, the lowest one was selected. If this was zero, no unit violations were considered for that complexity level. First, we observed that it can indeed be problematic to exclude dimensional analysis from the algorithm when the requirement for unit-adherent equations exists. This was reflected by median values of more than 80% of solutions with unit violations on the TD dataset and more than 40% on the FM datasets when the baseline algorithm without dimensional analysis was applied. The multi-objective approach not only found more solutions but also more solutions without unit violations. The difference was particularly drastic for the TD dataset, but could also be observed for the FM datasets. Evolutive culling and the repair mechanism ensured unit compliance of the equations, resulting in a final front with 0% of solutions with dimensional error. On this criterion, all proposed methods outperformed the baseline algorithm with statistical significance [224].

All unit-aware algorithms produced significantly fewer solutions with unit violations.

Unit-aware equations tended to use more constants. However, this difference is not always statistically significant.

The number of constants within an equation is an important quality criterion for domain experts, who prefer models with fewer constants. On the TD dataset, evolutive culling, and repair mechanism contained equations with significantly more constants in the PO front than the baseline algorithm. This could not be confirmed statistically for the FM datasets, but a similar tendency can be observed in the bottom left subplot of Fig. 8.2. Evolutive culling does not insert new constants into the tree like the repair mechanism does, but still shows higher usage of constants. This can be explained by the joker unit, which is introduced only by constants and propagated through the tree by most functions, encouraging the use of constants in equations. The multi-objective approach does not show significant differences to the baseline in the numbers of constants on all datasets.

The constraint handling methods had an impact on the algorithm runtime.

By looking at the number of generations completed within the time limit, we aimed to assess the runtime differences between the algorithms. The number of generations was normalized by the minimum number of generations a single run achieved within each dataset to account for different dataset sizes. It can be seen that evolutive culling tends to run more generations and the repair

mechanism runs fewer generations compared to the baseline algorithm. These observations were supported by the results of the statistical test. The runtime loss of the repair mechanism can be explained by the higher number of constants that needed to be fitted, which increased the duration of one generation. Although solutions with many constants may not appear in the final PO front due to their fitness, they remained part of the population throughout the evolutionary process. Evolutive culling excludes solutions with unit violations from the population, which led to smaller population sizes in the current implementation of the algorithm. This explains the higher number of generations performed by the algorithm. The multi-objective approach performed significantly more generations on the TD dataset compared to the baseline, which was not continued for the FM datasets. A more profound understanding of this behavior will require a closer look at the population dynamics during the evolution [224].

Difference between algorithms in the PO front mainly appeared at lower complexity values. They vanished for higher complexities.

Fig. 8.3 displays the 31 combined PO fronts for each algorithm on the TD dataset. We sought to examine the effects of the algorithms with dimensional analysis on the primary error objective MSE. Unsurprisingly, the baseline algorithm contained more solutions with unit violations in the PO front. Looking at the multi-objective approach, one can see that the unit-adherent solutions with complexities between five and eight had considerably higher MSE values than the ones with unit violations. The multi-objective approach filled gaps in the PO front with unit-violating solutions at lower complexities, which were not present in the PO fronts of evolutive culling and repair mechanism. These differences, however, almost vanished for higher complexities from nine to 15. For complexities above 16, all algorithms identified solutions with MSE values close to 0. The algorithms with dimensional analysis thereby had fewer unit violations than the baseline. While Fig. 8.3 only shows the TD dataset, similar observations were made for the FM datasets [224].

From the observations, we concluded that it was definitely beneficial to include unit information in the GP algorithm when the requirement for unit-adhering equations exists. The proposed algorithms revealed that accuracy was rarely compromised on the datasets used. From a practical perspective, we preferred the multi-objective approach as it offers decision makers multiple levels of unit violations per complexity. However, to better understand the strengths and weaknesses of each algorithm, further investigation on the population dynamics using more complex benchmark equations will be necessary [224].

8.4 Unit-Aware Equations for Particle-Laden Flows

To wrap up the advancements presented in this thesis, we applied the entire pipeline of algorithms, i.e., island models, MPNN as inductive bias for unit-aware GP, and inclusion of additional domain knowledge, to develop equations for the Stokes flow through arrangements of spherical particles. The goal was to identify symbolic models that approximated the MPNN with reasonable accuracy, but were significantly less complex. Additionally, these models had to adhere to physical laws.

8.4.1 Experiment Design

Stokes Flow Datasets

We performed experiments on the Stokes flow data published in [79], with volume fractions $\phi = \{0.05, 0.1, 0.2, 0.3, 0.4\}$. Training of the MPNN was performed on 1332 particle arrangements per volume fraction, and seven incremental rotations of each arrangement by $\frac{\pi}{4}$, resulting in 10656 arrangements. The dataset was split into training and validation sets with a ratio of 3:1, using a random split. The MPNN output of the validation data was used as training data for GP. This dataset contained 30 times the number of rows as the arrangements covered within the dataset, which equaled 79,920 rows. This was

because it learnt the messages sent between each of the 30 neighbors and the particle of interest, as approximated by the MPNN. These 79,920 samples were again randomly split into training and validation sets with a ratio of 3:1 for the GP training. The results reported in the following were assessed on a test set from different realizations of the simulation, which had not been seen during the training of both the MPNN and GP. The test data were also augmented by seven incremental rotations of each arrangement by $\frac{\pi}{4}$, leading to a total number of arrangements of 15,096 for $\phi = 0.05$, 7,104 for $\phi = 0.1$, 8,880 for $\phi = 0.2$, 1,776 for $\phi = 0.3$, and 2,664 for $\phi = 0.4$.

MPNN Settings

In our experiments, we employed the MPNN method presented in Sec. 7.1.1, using the underlying structure $y = g(x)$. This method had demonstrated similar accuracy to the nested function, while only utilizing the edge model. The MPNN settings introduced in Sec. 7.1.5 remain valid also for the experiments of this section. We performed six runs of MPNN training for each volume fraction, and selected the network with the lowest MSE on the test set as the basis for the subsequent SR.

GP Settings

To account for different problem complexities and to guarantee comparability, the GP algorithm was run for a fixed number of 150 generations rather than setting a time limit. We employed a population size of 500, and restricted the maximum model complexity to 35. The so-called Hall of Fame keeps track of an archive of Pareto-optimal solutions using the MSE, the dimension penalty, and the equation complexity as objectives. During evolution, the algorithm additionally considered the Spearman correlation as an objective, which had proven meaningful in the past in retaining promising solutions in the population. Based on the results of Chapter 6, we employed a multi-objective island model approach in the GP algorithm. To this end, the standard settings of `TiSR` were used, with four islands and five individuals migrating every fifty generations to randomly selected islands. The features and functions along with their complexity values and other algorithm settings are summarized in Tab. 8.5.

Summary of Domain Knowledge Integrated into the Algorithms

In the preceding sections, we illustrated how domain knowledge can be incorporated as a form of bias into the proposed machine learning pipeline at multiple levels. For the fluid mechanics problem, observational bias was introduced by converting the input data to spherical coordinate systems, thus saving the algorithm intermediate computation steps to compute relative locations of particles. Moreover, data augmentation by rotation of particle arrangements increased the number of available data samples, as well as the implicit representation of symmetries inherent to the problem. Inductive biases as defined earlier in this thesis were applied to shrink the search space of potential models by means that align with the nature of the problem. For the Stokes flow problem, the target variable was computed as a sum of pairwise interactions between particles, which were approximated by an MPNN as an inductive bias for GP. The GP algorithm itself employed a learning bias to encourage certain characteristics of the final model. First, some operations such as trigonometric functions were prohibited from nesting, to avoid overfitting the training data and promote human-interpretable equations. Second, constants and non-linear operations were assigned a higher complexity value of two to keep the algorithm from excessively using them with diminishing returns in terms of accuracy. Third, the unit-aware algorithms assessed in the first parts of this chapter were employed, with a dimensional analysis that accounts for new constants with undetermined units. Among the constraint handling methods evaluated previously, we selected the multi-objective approach with a dimension penalty as an additional objective. This had previously demonstrated the capacity to offer more solutions at lower complexity levels, despite the presence of unit violations. In addition to the biases summarized in Tab. 8.5, we would like to point out that selecting an appropriate set of features and functions is another essential way of integrating domain knowledge, which is, however, not listed here as it is already inherent to the GP algorithm.

Table 8.5: Algorithm configurations.

Parameter	Settings
Population size	500
Max. complexity of equations	35
No. generations	150
Optimization objectives	MSE, Spearman correlation, dimension penalty, complexity
Hall of Fame objectives	MSE, dimension penalty, complexity
Training features	r [m], θ [0], φ [0]
Target feature	Force F [N]
Function set	$+, -, \cdot, \div, e^{\circ}, \log(\circ), \sin(\circ), \cos(\circ), \circ^{\circ}$
Complexity values	
Features	1
Constants	2
$+, -, \cdot, \div$	1
$\circ^{(\circ)}$, $\sin(\circ)$, $\cos(\circ)$, e°, $\log(\circ)$	2
Implementation	`TiSR` [170]

Table 8.6: Overview of domain knowledge integrated as bias into the algorithms to learn unit-aware equations for the variation from the mean drag in Stokes flow.

Bias Type	Bias
Observational	Conversion of input data from Cartesian to spherical coordinate system
Observational	Augmentation of training data for implicit representation of symmetries in the dataset
Inductive	MPNN as inductive bias to approximate the underlying pairwise interactions between particles and subdivide the problem
Learning	Prohibiting nestings of trigonometric, logarithmic and exponential functions
Learning	Higher complexity values for constants and non-linear operations
Learning	Dimensional analysis including new constants as optimization objective to promote unit-aware equations

8.4.2 Results and Analysis

Selection Procedure

In the following, we provide an analysis of the resulting equations regarding their building blocks used, as well as the accuracy and number of constants per equation. While it can be meaningful for decision makers to look at the entire PO front, we focused on the unit-aware equations without dimension violations. For each of the 31 runs, we selected the three least complex equations per run within the top 95% in terms of R^2 on the test set, summing up to 93 equations per volume fraction. These were manually scanned for repeating building blocks, and a few representative equations are reported in Tab. 8.7, covering different building blocks identified, as well as multiple complexity and R^2 levels.

The identified building blocks aligned with the results of previous research and with the expected behavior of the functions.

All equations in Tab. 8.7 showed a dependency on θ and r, and did not include the azimuthal angle ϕ. The polar angle θ only occurred as an argument of sine and cosine functions. Repeating building blocks in this regard were $\sin(\theta)$, $\sin^{\circ}(\theta)$ and $\cos(2\theta)$ or $\cos(\circ \cdot \theta)$, where the coefficient was close to 2 for the latter. These building blocks were in accordance with the ones identified in our related studies [79, 223]. Only $\sin^{\circ}(\theta)$ was new in these experiments and deserves further attention in the future. Almost all equations incorporated the radius r multiple times and in various ways, which was also observed in the equations presented in Sec. 7.1.6. Common shared building blocks were $\frac{1}{r}$, $\exp^{-r \cdot \circ}$, and $\frac{1}{r^{\circ}}$ with varying exponents in the denominator. This aligned with

ϕ	Equation	Compl.	GP R^2	MPNN R^2
0.05	$\frac{-1.7335+\sin^{2.0134}(\theta)r}{5.0816+r^3}$	21	0.799	0.802
	$\frac{0.12016(-3.4178+r-r\cos(2\theta))}{r^2}$	17	0.767	
	$0.13866\,(6.098-r)\left(-0.80455+e^{\frac{-1.3913+\sin(\theta)}{r}}\right)$	24	0.796	
0.1	$0.016433+(-0.19197+0.052522r)\left(\frac{1.2703}{r}+\cos(2\theta)\right)$	22	0.767	0.763
	$\left(-0.986+r\left(-0.16147+\sin^{2.2623}(\theta)\right)\right)e^{-1.2618r}$	22	0.761	
	$(-0.79563+r\,(-0.40474+\sin(\theta)))\,e^{-1.1265r}$	18	0.748	
0.2	$\frac{0.57466(-0.28057-0.64812r+r\sin(\theta))\left(-1.9911-r^2+3.4073r+\sin(\theta)\right)}{r^3}$	30	0.764	0.774
	$\left(-0.22677+\frac{0.7755}{r}\right)\left(\frac{-0.37034}{r}-\log(1.6099-\sin(\theta))\right)$	21	0.725	
	$0.081558+\frac{-0.2043}{r}+(0.28823-0.10502r)\sin(-64.414+2\theta)$	25	0.757	
0.3	$\frac{0.060991r+0.084136r^3-0.2317r^2+\left(1.3462-2.2383r+0.68761r^2\right)\cos(2\theta)}{r^3}$	29	0.692	0.691
	$\frac{-0.6663+\sin^{2.0254}(\theta)}{2.2286+0.086684r^{6.7198}}$	22	0.660	
	$\frac{-0.78737+\sin(\theta)}{1.6711+0.067261r^{6.7192}}$	18	0.653	
0.4	$\frac{\left(0.58018+0.51954r^2-1.3825r\right)(-0.34309+0.58373r+r\cos(-2.037\theta))}{r^3}$	27	0.685	0.701
	$(0.067115-0.097299\,(-3.5636+r)\cos(-2.0365\theta))\,(-2.1899+r)$	22	0.674	
	$\frac{\left(-0.28042-0.43622r^3+0.99567r^2\right)(0.34516-0.58526r-r\cos(-2.037\theta))}{r^4}$	29	0.688	

Table 8.7: Symbolic models and the GP and MPNN performance on the test dataset. All equations are aggregated over 30 neighboring particles, with the sum symbol omitted here for clarity.

the known behavior of these functions that decay with increasing r, as reported in [79, 223].

The resulting equations and the MPNN had similar R^2 values.

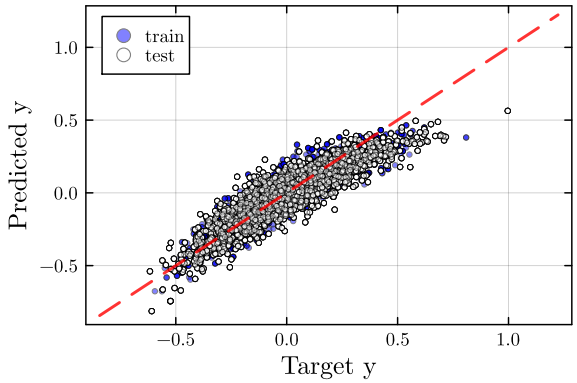
Tab. 8.7 moreover reports the performance of different equations in terms of R^2, together with the respective MPNN performance. The values were computed on a test dataset generated from simulation runs that had not been seen during training. Overall, the accuracy of the MPNN was very similar or slightly better compared to the symbolic models. Both model types showed a decreasing accuracy with increasing volume fraction, ranging from approximately 0.8 for $\phi = 0.05$ to 0.68 for $\phi = 0.4$. At the same time, the symbolic models were by orders of magnitude less complex than the GNN and offered insights into the underlying relations. Fig. 8.4 displays the correlation plot between the target and predicted values for the most accurate symbolic models, i.e., the first for $\phi = 0.05$ and third for $\phi = 0.4$. The R^2 values for training and testing were very similar, indicating no overfitting to the training data.

Considerations on Number of Constants

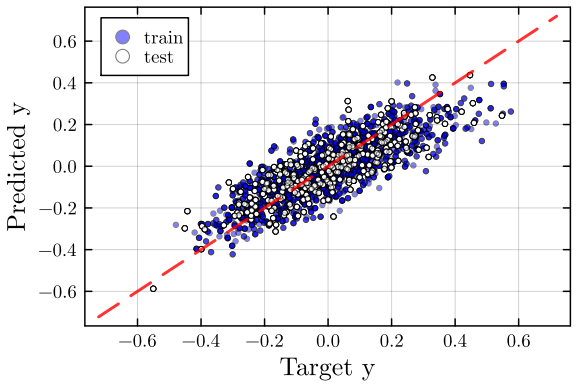
The benefit of the equations presented here was their dimensional consistency, i.e., potential unit violations were effectively balanced by constants. This can be achieved with the intelligent placement of constants, or with the use of more constants. Our algorithm did not explicitly minimize the number of constants, but constants were given a higher complexity value of two, and the complexity was minimized. We observed an increasing number of constants in the equations with increasing volume fraction. Fig. 8.5 displays a boxplot for the number of constants within the 93 selected best solutions per volume fraction. We can indeed see that the equations for $\phi = 0.05$ and $\phi = 0.1$ exhibited a median of four constants, which increased to six constants for the other volume fractions. This can be explained by the increased problem complexity for higher volume fractions, which was also reflected in the R^2 values in Tab. 8.7. The selection method used in this section, i.e., the top 95% of the equations within a Pareto front, might have also been a reason. Good equations are thus more complex for higher volume fractions, which nevertheless did not show signs of overfitting

Figure 8.4: Plot of the target values against the predicted values for different volume fractions on training and test datasets.

(a) $\phi = 0.05$, with R^2 of 0.826 on the train and 0.799 on the test set.



(b) $\phi = 0.4$, with R^2 of 0.675 on the train and 0.688 on the test set.



to the training data. However, further investigation will be necessary to better understand the underlying factors contributing to this increased complexity and its implications for the generalizability of the equations.

8.4.3 Comparison to Baseline

Comparison to Chapter 5

A major goal of this thesis is to demonstrate the inclusion of domain knowledge in GP algorithms and their application to problems with scalable complexity and unknown ground truth. Unlike the Stokes flow past two spherical particles discussed in Chapter 5, the current study expands to systems involving numerous particles in various configurations. Moreover, the focus was shifted from predicting the full velocity field around particles to directly predicting the deviations from the mean drag given the relative locations of the surrounding neighbors. In this context, only a limited number of neighboring particles were considered. In the experiments presented in this section, this was set to 30 to balance between computational feasibility and physical accuracy. The performance of the equations evolved by GP in Tab. 8.7 was comparable to that of the ANN-based methods in terms of R^2. This demonstrates a significant improvement compared to the results in Tab. 5.4 and Fig. 5.3, where the MLP outperformed the evolved equations by multiple orders of magnitude. Considering the increased problem complexity involving more particles, the use of MPNNs as an inductive bias for GP proved to be well-suited for predicting the deviations from the mean drag in the Stokes flow.

Comparison to Chapter 7

Compared to Chapter 7, the experiments performed in this section employed more realistic and complex datasets with particle arrangements not only in free stream but also in the presence of a theoretically infinite number of particles. The derived expressions in Tab. 8.7 exhibited a large overlap with the expressions and building blocks identified in Sec. 7.1.6 and remained dimensionally

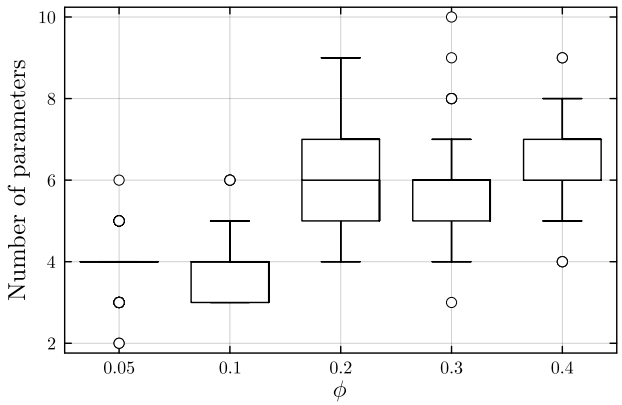


Figure 8.5: Number of constants in the three least complex equations within the top 95% R^2 range per run.

consistent. This consistency further validated the proposed method, as the introduction of dimensional constraints did not reduce the predictive accuracy of the expressions, and simultaneously satisfied unit constraints.

Impact Within the Research Area of the Problem Domain

In [79], the authors presented a unified expression to predict the deviation from the mean drag in Stokes flow. It was manually constructed from building blocks that had previously been evolved by GP, using an MPNN as inductive bias as in the proposed ML pipeline of this thesis. When compared to this universal equation with different coefficients fitted for each volume fraction, the equations in Tab. 8.7 were slightly less concise and less accurate. Moreover, the unified expression only used four coefficients, whereas the expressions evolved here tended to use more coefficients with increasing volume fraction. This can be partially attributed to the unified equation accounting for twice as many neighboring particles, which inherently offers an advantage in precision. However, further investigation is required to understand how the number of neighbors considered affects the equations evolved by GP. Within the broader area of related work, this approach was the first to offer symbolic models for the prediction of the deviations from the mean drag in Stokes flow, with accuracy values comparable to those of state-of-the-art approaches. Overall, the ability to adapt the algorithm and incorporate bias on multiple levels demonstrated its flexibility and potential for broader applicability.

8.5 Unit-Aware Equations for the Inverse Kinematics Problem

Similar to the previous section, we aimed to replace the network blocks in the MPNN to approximate the IK problem with symbolic models. Due to the nested underlying function $y = f(g(x))$ employed for this problem, multiple equations were learned to replace each element in the message vector and to approximate the joint angles θ_i using these messages as additional input features.

8.5.1 Experiment Design

Inverse Kinematics Datasets

The MPNN was trained on 4.5 million samples for 3 DOF and 9 million samples for 5 DOF, with the data split randomly into training and validation sets in a 4:1 ratio. The model predictions on the validation dataset, comprising 900,000 samples for 3 DOF and 1.8 million samples for 5 DOF, were used as inputs for the GP. Two types of equations were learned with GP: first, the message equations, each predicting one of the six elements in the message vector within the MPNN, and second, the node equations, predicting the joint values to reach a certain pose. The dataset size increased for the message equations, with 4 messages sent per node for 3 DOF, resulting in 3.6 million rows, and 8 messages for 5 DOF, leading to 14.2 million rows. However, GP frameworks for symbolic regression are typically not optimized to handle such large-scale

Table 8.8: Algorithm configurations for the final IK experiments.

Parameter	Settings
No. generations	200
Max. complexity of equations	40
Training features for m_i	x [m], y [m], z [m], Φ [0], Θ [0], Ψ [0], α_s [0], $\theta_{s,\text{off}}$ [0], $\theta_{s,\text{ref}}$ [0], l_s [m], α_r [0], $\theta_{r,\text{off}}$ [0], $\theta_{r,\text{ref}}$ [0], l_r [m]
Training features for θ_i	x [m], y [m], z [m], Φ [0], Θ [0], Ψ [0], α_i [0], $\theta_{i,\text{off}}$ [0], $\theta_{i,\text{ref}}$ [0], l_i [m], m_1 [0], m_2 [0], m_3 [0], m_4 [0], m_5 [0], m_6 [0],
Function set	$+, -, \cdot, \div, e^{\circ}, \log(\circ), \sin(\circ), \cos(\circ), \circ^{\circ}$
Complexity values	
Features	1
Constants	2
$+, -, \cdot, \div$	1
$\circ^{(\circ)}$, $\sin(\circ)$, $\cos(\circ)$, e°, $\log(\circ)$	2
$\arctan 2(\circ, \circ)$, $\arccos(\circ)$	2

datasets efficiently. To ensure that the experiments could be completed within a reasonable timeframe of up to three days per run, we limited the dataset to 10,000 samples per link length configuration, resulting in a total of 90,000 samples. For the message vectors, the reduced dataset thus corresponded to 360,000 rows for 3 DOF and 720,000 rows for 5 DOF. These were split in a 4:1 ratio into a train and a validation set. The node features employed the aggregated incoming messages per node as additional features, so that the node equations were trained on 90,000 samples each. These were again randomly split into train and validation sets, with a 4:1 ratio. Additionally, a test set of 10,000 samples was employed with a previously unseen combinations of link lengths within the reachable area available during training.

MPNN Settings

In the following experiments, we employed the method presented in Sec. 7.2.1, with an underlying nested function $y = f(g(x))$. The same algorithm settings as proposed in Sec. 7.2.3 were applied. We selected the same models that were previously assessed as the best models in Sec. 7.2.4.

GP Settings

We kept the GP settings similar to the ones used in the fluid mechanics experiments (see Tab. 8.5). A change was made in terms of the performed number of generations, which were set to 200 to account for the high problem complexity, but otherwise stuck to the settings employed in the preliminary experiments in Sec. 4.4. The maximum allowed complexity of an equation was set at 40. Moreover, the sets of features and functions were modified to match the IK problem. It should be noted that the set of features for 5 DOF was modified according to the insights in terms of feature importance presented in Fig. 7.14. For θ_1 and θ_2, the features x and y were removed; for θ_3, θ_4, and θ_5, the features x, y, and z were removed. The set of potential functions was selected in collaboration with the domain experts to ensure a minimal yet comprehensive selection. This approach aims to prevent an unnecessary increase in the search space while still encompassing all potentially significant functions. Consequently, the $\arcsin(\circ)$ function was excluded as it can be derived from $\arccos(\circ)$, and $\tan(\circ)$ was omitted since it can be computed using $\sin(\circ)$ and $\cos(\circ)$.

Summary of Domain Knowledge Integrated into the Algorithms

Throughout the previous sections, different types of domain knowledge were incorporated as a form of bias into the proposed machine learning pipeline for the IK problem. Tab. 8.9 summarizes the biases employed for the final IK experiments. The reference-guided approach as proposed by the domain experts introduced observational bias by employing a reference angle close to the target joint angle in the training features. This approach was inspired by robot path planning, where the path between two poses is divided into multiple intermediate poses, which are close to each other. It was assumed that the

Table 8.9: Overview of domain knowledge integrated as bias into the MPNN and GP algorithms to approximate the variation from the mean drag in Stokes flow.

Bias Type	Bias
Observational	Reference-guided approach with a reference angle θ_{ref} in the training data
Inductive	MPNN as inductive bias to approximate the pairwise interactions between joints and subdivide the problem
Learning	Prohibiting nestings of trigonometric, logarithmic and exponential functions
Learning	Higher complexity values for constants and non-linear operations
Learning	Dimensional analysis including new constants as optimization objective to promote unit-aware equations
Learning	Modified computation of distances between angles using central angles in the fitness function

msg	Equation	Compl.	R^2
1	$-2.5122 + 0.53\frac{\theta_{s,\text{off}}}{\Psi} + 0.53\frac{\Phi}{\Psi} + 1.06\theta_{s,\text{off}} + 1.06\theta_{r,\text{ref}}$	17	0.808
2	$-2.16 + 1.39\,(\alpha_r + \theta_{r,\text{ref}}) - 0.161\alpha_r\alpha_s - 0.322\alpha_r\theta_{r,\text{off}} - 0.161\alpha_s\theta_{r,\text{ref}} + 0.17236\Psi\theta_{r,\text{ref}} - 0.322\theta_{r,\text{off}}\theta_{r,\text{ref}} - 0.019964\alpha_s\Psi\theta_{r,\text{ref}} - 0.039928\Psi\theta_{r,\text{off}}\theta_{r,\text{ref}}$	23	0.828
3	$\frac{1.41}{\Psi} + 3.62\frac{\theta_{s,\text{off}}}{\Psi} - 0.885\frac{\alpha_r\theta_{r,\text{ref}}}{\Psi} - \frac{\theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi} + 4.77\alpha_r - 2\theta_{s,\text{off}} - \alpha_r\theta_{r,\text{ref}}$	27	0.754
4	$-0.17456\theta_{r,\text{ref}}^2 - 1.18\theta_{s,\text{off}} + 1.7979 l_r\theta_{r,\text{ref}} - 0.34911 l_r\Psi\theta_{r,\text{ref}}$	18	0.611
5	$-0.275\alpha_s - \theta_{r,\text{ref}} + 1.023\theta_{s,\text{off}} + 2\theta_{r,\text{off}} - 0.55\theta_{s,\text{off}}\theta_{r,\text{ref}} + 0.3432\theta_{s,\text{off}}\Phi$	20	0.846
6	$-0.2683 - 2.58\frac{\theta_{s,\text{off}}}{\Psi} + \frac{\theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi} + 1.104\theta_{r,\text{ref}} - 1.672\theta_{s,\text{off}} - 2\theta_{r,\text{off}} + 0.648\theta_{s,\text{off}}\theta_{r,\text{ref}}$	20	0.874

Table 8.10: Symbolic models that replace the six elements in the message vector of the MPNN for 3 DOF manipulators.

distance in joint angles between consecutive intermediate poses was also small. Consequently, the reference angle represented the manipulator configuration at the preceding pose along the path. Inductive bias was introduced by approximating pairwise interactions between neighboring joints within the kinematic chain using an MPNN. The aggregated messages were employed as additional features in the prediction of the target joint angle. The learning bias in terms of function nesting, custom complexity values and unit-awareness implemented in the GP algorithms remained the same as for the fluid mechanics problem. Moreover, we introduced learning bias in the distance computation between the target and predicted joint angles in the fitness function, which was modified to incorporate central angles. The goal was to compute small distances between angles that were close to each other on the unit circle, avoiding large deviations when calculated with a simple subtraction due to different rotation directions.

8.5.2 Results and Analysis

Selection Procedure for Message Equations

For the equations replacing the six elements in the message vector, we selected the three least complex equations within the top 10 percent in terms of R^2 in the Hall of Fame for each of the 31 runs. These 93 equations were then manually reviewed in collaboration with the domain expert for meaningful or repeating building blocks. Moreover, we looked at the Pareto fronts to identify potential knee points and select a representative equation that balanced accuracy and complexity. The corresponding Pareto fronts are displayed in Fig. 8.6 for the 3 DOF manipulator, and in Fig. 8.7 for the 5 DOF manipulator. The selected equations are shown in Tabs. 8.10 and 8.11 respectively, together with their complexity and R^2 values. To gain more in-depth insights into the building blocks within the equations, they are displayed in a simplified and expanded way using the `julia` package `SymbolicUtils.jl` and its expression manipulation functions. This may have led to inconsistencies between the

msg	Equation	Compl.	R^2
1	$0.46 + \alpha_s - 0.712\Theta - 1.024\Theta\theta_{s,\text{off}} + \cos(\alpha_r + \alpha_s)$	20	0.669
2	$0.343 - 0.16298\alpha_r - 0.32597\theta_{s,\text{off}} - 0.59 l_s + 0.12585\alpha_r \arctan(\Psi, \theta_{s,\text{off}}) + + 0.2517\theta_{s,\text{off}} \arctan(\Psi, \theta_{s,\text{off}}) + 0.45558 l_s \arctan(\Psi, \theta_{s,\text{off}})$	24	0.500
3	$-0.73839 + 0.302\Psi - 0.18573\theta_{s,\text{off}} - 0.302\theta_{s,\text{off}} \arctan(0.01728\Psi\theta_{s,\text{off}}, 0.000667 - \alpha_r)$	25	0.520
4	$\alpha_r - 3.63 l_s + 0.754 \arctan(3.07 - 2\theta_{s,\text{off}}, 0.0153 - 2\Psi)$	20	0.670
5	$-0.89357 + 0.2008\alpha_r^2 + 0.30292\Theta + 0.40161\theta_{s,\text{off}} + 0.55582\Theta\theta_{s,\text{off}}$	21	0.642
6	$0.834\theta_{s,\text{off}} - 2.3769\alpha_r + 3.5178 l_r + 1.4095\theta_{r,\text{off}} - 0.834\theta_{r,\text{off}} \arctan(\theta_{r,\text{off}}, \Psi)$	24	0.566

Table 8.11: Symbolic models that replace the six elements in the message vector of the MPNN for 5 DOF manipulators.

length of the displayed equation and the complexity value indicated in the tables, which refers to the complexity of the originally evolved equation. A larger selection of these equations can be found in Appendix A.

The message equations exhibited structural differences between runs, but shared a few building blocks.

It can be observed that the message equations for 3 DOF achieved higher accuracy values compared to the 5 DOF equations. This may be attributed to the two additional hidden layers in the network blocks for 5 DOF, enabling the network to compute more complex outputs. Since the target variables of the message elements were approximated by the MPNN, the ideal solution and the specific computations within the message equations are unknown. Therefore, interpreting these equations is a challenging task, further amplified by structural differences in the equations observed across multiple runs. Despite these differences, a few building blocks appeared multiple times. For 3 DOF, the term $\frac{\circ}{\Psi}$ could be found in several equations, sometimes multiple times per equation. In the numerator of this term, often variables with only a few distinct values in the input space were located, such as θ_{off} or α. We interpreted this behavior as an attempt by the algorithm to differentiate between cases for the manipulator or between joints, such as distinguishing whether the elbow was positioned up or down to reach a specific pose. Moreover, the reference angle of the receiving joint $\theta_{r,\text{ref}}$ appeared in all equations. Information from the sending joint was mainly incorporated in the form of the angles $\theta_{s,\text{off}}$ and α_s, which again took on a limited set of values and were often equal to zero. For the 5 DOF message equations, we first observed the absence of reference angles, and an increased occurrence of the link length of the sending joints l_s, as well as $\theta_{s,\text{off}}$ and Ψ. The missing reference angles could be explained by the fact that this was the main feature used in the node model according to the feature importance analysis in Figs. 7.12 and 7.14. Consequently, the other features became more significant within the message models.

The output of the selected message equations was computed and served as additional training features for the node equations. As a result, any errors in computing the message elements may have carried over and affected the accuracy of the node equations.

A perfect equation for θ_1 in 3 DOF was discovered by the algorithm.

We followed a similar selection procedure as for the message equations, which was, however, drastically simplified since most node equations exclusively relied on the reference angle. Tabs. 8.12 and 8.13 display two representative equations per joint, together with their respective R^2 and MSE values. First, these values raised an important concern about the suitability of R^2 for selecting equations, as the small changes in R^2 (e.g., 0.002 between θ_1 and θ_2) could have corresponded to a substantial increases in MSE. Aside from this, we could see that the algorithm found a perfect fit for the first joint in 3 DOF. Due to the manipulator configuration, the first joint could be directly inferred from the Φ angle of the target orientation. The other orientation feature Ψ took on the value $\frac{\pi}{2}$ or $-\frac{\pi}{2}$, and accounted for bend overs of the manipulator to reach the final pose. The constant 1.57 corresponded to the value of $\frac{\pi}{2}$. It is to be

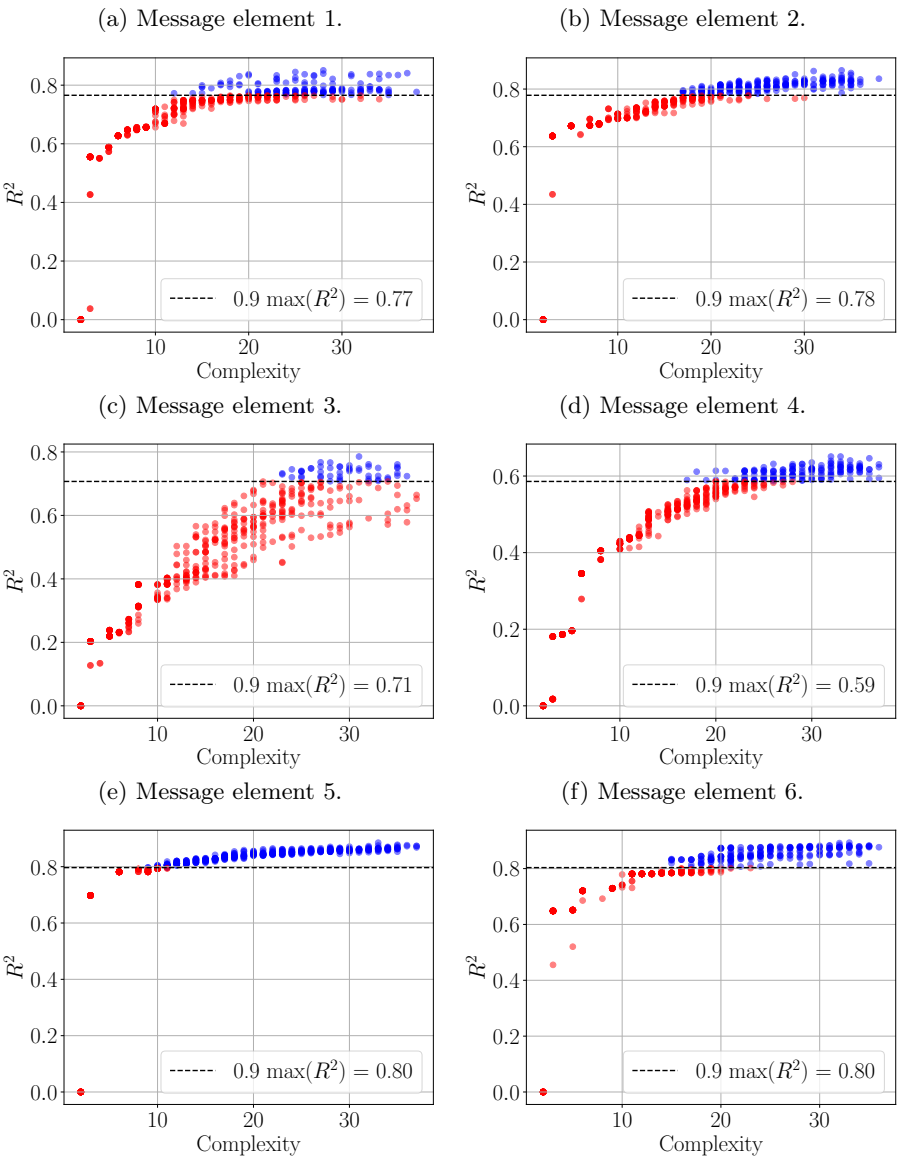


Figure 8.6: Pareto-optimal fronts for the six message elements combined over 31 runs for 3 DOF. The blue solutions above the dashed line are the top 10 percent of solutions in terms of R^2.

noted that the term $\arccos(\Psi)$ only produced meaningful results here because its input values were clipped to zero or one in case they exceeded this range, with $\arccos(0) = \frac{pi}{2}$ and $\arccos(1) = 0$.

The other node equations mainly relied on the reference angle, and failed to learn the underlying physics.

For θ_2 and θ_3 of 3 DOF and all the joints of 5 DOF, the term $0.997\theta_{\text{ref}}$ approximated the target variable best, with an MSE of about 0.01. The equation $(\theta_{\text{ref}} \cdot \theta_{\text{ref}})^{0.499}$ also appeared frequently in the final Hall of Fame, which was, however, more complex. The message elements that were intended to encode relationships between joints were not used in the majority of the equations, with the exception of θ_2 of the 3 DOF manipulator, which incorporated msg_6 frequently. However, the increased complexity of this equation did not result in a significant improvement in MSE. Considering the feature importance analysis in Fig. 7.14, a high reliance of the node equations on the reference angles was expected for 5 DOF. Contrary to expectations, the MPNN for 3 DOF revealed only a marginal importance of the reference angle in the node model in Fig. 7.13. However, this observation could not be confirmed for the 3 DOF equations for θ_2 and θ_3, which employed the reference angle as the most important feature. This indicates that the underlying physics for 3 DOF were approximated well by the MPNN, whereas the GP algorithm was unable to learn these relationships effectively. We attributed this to the inclusion of the reference angle as an additional feature, which already approximated the target angle with a certain accuracy and thus introduced a strong local optimum

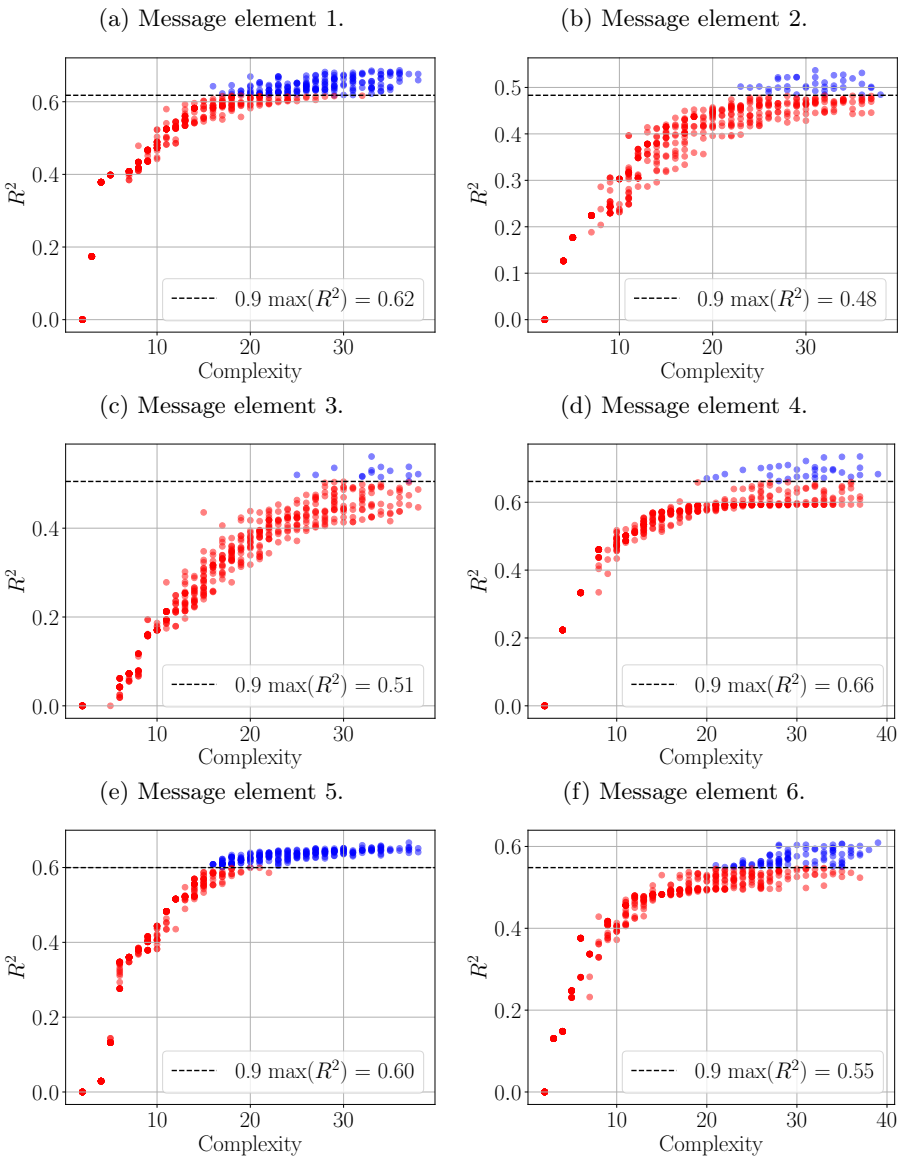


Figure 8.7: Pareto-optimal fronts for the six message elements combined over 31 runs for 5 DOF. The blue solutions above the dashed line are the top 10 percent of solutions in terms of R^2.

within the dataset. Escaping this local optimum might have been difficult for the algorithm and potentially hindered the learning of the underlying physics. The learning power of the MPNN was probably sufficient to find a solution outside this local optimum, whereas the GP algorithm could not achieve the same. A potential reason for this could have been the usage of ReLU as an activation function in the MPNN, which enables the network to deactivate certain computations when the input is below zero. This ability to selectively activate or deactivate certain pathways was not available in GP, but it can help to distinguish between cases and resolve ambiguities. Consequently, GP had to find other ways to encode such distinctions within the evolved equations, which were, however, not present in the learned node equations. Overall, the learned node equations were not useful in practice. Nevertheless, our experiments highlighted that including domain knowledge can, in the worst case, misguide the algorithm when the boundaries are set too narrowly.

8.5.3 Comparison to Baseline

Comparison to Chapter 4

Compared to Chapter 4, the problem modeling underwent some modifications; however, we performed a comparative analysis to assess the effects of these changes. In the baseline approach, relationships within the kinematic chain were modeled using cooperative coevolution, combined with a mechanism to reverse the influence of preceding joints within the training data for subsequent

Table 8.12: Symbolic models that predict the target angles of each joint to reach a specific pose for 3 DOF manipulators.

node	Equation	R^2	MSE	Compl.
1	$1.57 - (\Psi - \Phi)$	0.999	1.622e-14	6
1	$\Phi - \arccos(\Psi)$	0.999	1.622e-14	5
2	$14.0 \arctan(-30.8, \text{msg}_6)$	0.997	0.0110	8
2	$0.997\theta_{\text{ref}}$	0.997	0.0111	4
3	$(\theta_{\text{ref}} \cdot \theta_{\text{ref}})^{0.499}$	0.996	0.0111	7
3	$0.997\theta_{\text{ref}}$	0.997	0.0112	4

Table 8.13: Symbolic models that predict the target angles of each joint to reach a specific pose for 5 DOF manipulators.

node	Equation	R^2	MSE	Compl.
1	$(\theta_{\text{ref}} \cdot \theta_{\text{ref}})^{0.499}$	0.997	0.0111	7
1	$0.997\theta_{\text{ref}}$	0.997	0.0112	4
2	$(\theta_{\text{ref}} \cdot \theta_{\text{ref}})^{0.499}$	0.997	0.0111	7
2	$0.997\theta_{\text{ref}}$	0.997	0.0112	4
3	$0.998(-0.00163 + \theta_{\text{ref}})$	0.997	0.0111	7
3	$0.997\theta_{\text{ref}}$	0.997	0.0112	4
4	$\frac{0.0631}{\arctan(0.0632, \theta_{\text{ref}})}$	0.997	0.0111	8
4	$0.997\theta_{\text{ref}}$	0.997	0.0112	4
5	$(\theta_{\text{ref}} \cdot \theta_{\text{ref}})^{0.499}$	0.997	0.0111	7
5	$0.997\theta_{\text{ref}}$	0.997	0.0111	4

joints. Here, we employed an MPNN to approximate pairwise relationships between joints, and aimed to divide the problem into smaller portions manageable for GP. Results of preliminary experiments in [203] indicated an improved performance when a reference angle was given as an additional training feature, as inspired by robot path planning. The primary motivation for this change in problem representation was to mitigate the multimodal nature of the problem, which had also led to problems in the IK-CCGP-3 experiments in Chapter 4. In essence, the reference angle provides guidance to the algorithm, indicating whether the manipulator elbow should point up or down to achieve a given pose. While this approach is both straightforward and intuitive, it introduces strong local optima in the search space, which hinders the development of equations representing the underlying physics properly. The cooperative coevolutionary approach also encountered the challenge of local optima, but at a different level. Instead of being introduced at the dataset level, these optima arose during the simultaneous training of three joints. Errors in one joint propagated through the kinematic chain, amplifying their effects on subsequent joints. From these observations, it can be concluded that more effective strategies are required to address the multimodality inherent in the IK problem when tackled with GP.

Comparison to Chapter 7

In addition, we compare the performance of the equations evolved with GP to the MPNN in Sec. 7.2.4. To this end, we selected the equation $0.997\theta_{\text{ref}}$ to approximate all the joints of 3 and 5 DOF manipulators, since it was the least complex equation with a reasonable MSE. For the 3 DOF manipulator, the first joint was predicted by $\Phi - \arccos(\Psi)$. The predicted joint angles were fed to the FK equations to compute the resulting pose. Fig. 8.8 displays the distributions of position and orientation errors on the same validation and test sets as employed in Sec. 7.2.4. The equations exhibited similar performance on both the validation and test sets, which was expected, as they all depended on the reference angle, which followed the same distribution across both datasets. Comparing these results to Fig. 7.10, it became apparent that the position error for 3 DOF was more than three times higher, with a mean error of 0.03553 ± 0.02185m. A similar trend was observed for the orientation

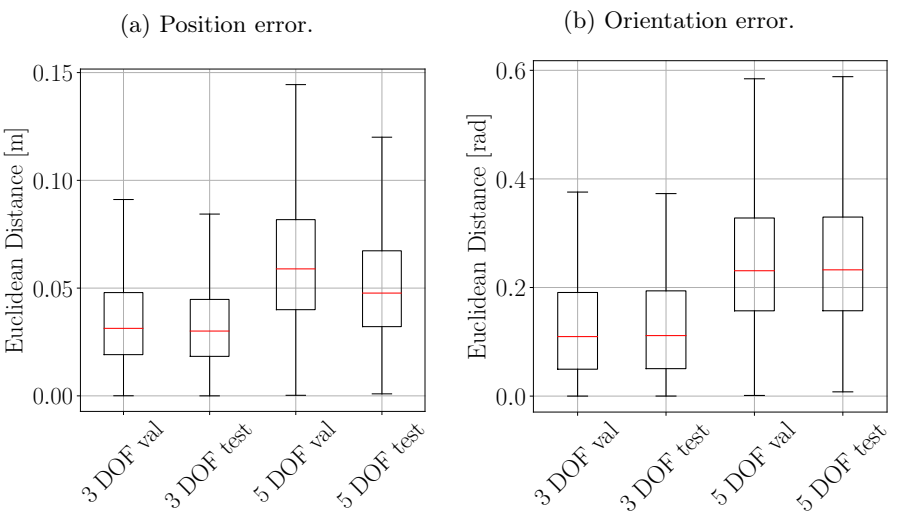


Figure 8.8: Distributions of position and orientation errors of the equations evolved with GP for 3 and 5 DOF manipulators on the validation and test datasets.

error of 0.29225 ± 0.85113rad. For the 5 DOF manipulators, an increase in error was also observed, although it was less severe, as the error in the MPNN prediction was already higher compared to the 3 DOF predictions.

Contributions to the Research Area of the Problem Domain

At this stage, and considering the inconsistent results obtained using MPNNs as inductive bias for GP on the IK problem, the developed equations are not useful for practical applications. However, we showed how an intuitive approach to including a reference angle to overcome the multimodality of the problem can misguide the algorithm. Despite the challenges, we consider these partially failed attempts to offer valuable insights for future research in this area. Overall, this approach is still a work in progress, and we aim to evaluate its performance with improved mechanisms to address the multimodality of the problem. To this end, more detailed domain knowledge is required.

8.6 Discussion and Limitations of the Proposed Methods

Throughout this thesis, we have proposed and studied several algorithms using two problems from robotics and fluid mechanics. While these problems differed in their specific application area, they share characteristics frequently encountered in science and engineering, and have not yet been fully understood by humans. To wrap up the algorithmic advances presented in this chapter, we review the results of the unit-aware GP approach, and critically discuss the advantages and potential limitations of the overall algorithmic pipeline.

Considerations on the Proposed Dimensional Analysis for Unit-Aware GP

The unit-aware GP approach considers constants with unknown units as jokers in the dimensional analysis. In our experiments, we did not observe disadvantages in terms of accuracy when unit-aware GP was employed, and we believe that within the multi-modal search space of equations, this algorithm guides the search towards those solutions that are also physically meaningful and interpretable. Nevertheless, since the constants were adjusted *a posteriori* to balance unit violations, it was possible for them to adopt unconventional or non-physical combinations of units. This could have resulted in the development of equations that were mathematically valid but physically questionable, undermining the interpretability and applicability of the resulting models. Therefore, to enhance both the interpretability and the physical consistency of the resulting equations, it may be advantageous to define a problem-specific set of plausible balance units for these constants in advance.

The proposed ML pipeline is suitable for the problem of particle-laden flows.

For the problem of particle-laden flows, using MPNNs as an inductive bias for unit-aware GP has proven to be a viable method. We successfully developed equations that adhered to physical laws while achieving accuracy comparable to the predictions of the MPNN. Moreover, this method shows potential for

extension to more complex problems, such as those involving higher Reynolds numbers, to better understand its limitations. While ANNs have previously demonstrated the ability to approximate higher Reynolds numbers well [255], there is currently no research available on symbolic models capable of achieving the same.

We attribute the issues of the ML pipeline on the IK problem more to the problem representation with a local optimum introduced by the reference angle, than to the approach itself.

The limitations of the ML pipeline under consideration became more obvious when applied to the IK problem. Combining MPNNs and GP introduces a multi-stage learning pipeline. In the first stage, the joint angles predicted by the MPNN resulted in a practically acceptable position and orientation error for 3 DOF. However, the errors for 5 DOF were larger, and its practical applicability would depend on the specific use case and should be assessed by domain experts. In the second stage, the symbolic models fitted to the recorded messages within the MPNN already showed deviations from the true messages. The node models predicting the joint values did not make use of the additional message features as expected, and almost fully relied on the reference angle. Two major issues with the presented ML pipeline became obvious in these conducted experiments. First, the error in the MPNN predictions can propagate further to the GP algorithm, potentially amplifying the resulting prediction error. Likewise, the approximation of message elements with a symbolic model may introduce errors in the additional training features for the node models. Second, MPNNs are not inherently designed to handle multimodal problems effectively. When used as inductive bias, they may fail to provide useful guidance for GP in such scenarios.

Considerations on Influence of Domain Knowledge for the IK Problem

Our experiments demonstrated how incorporating domain knowledge into the problem representation can influence the success of an algorithm. Despite their impracticality, the equations evolved with GP revealed the impact of the local optimum introduced by the reference angle, and motivated us to reconsider its usefulness in general, also for the MPNN. For this specific case, we claim that additional knowledge about both kinematics and algorithms is required to carefully anticipate how knowledge can guide the algorithm towards meaningful results, and avoid certain pitfalls. Possible approaches to tackle the multimodality inherent to the IK problem include an implicit case differentiation through filtered datasets, limiting the potential joint ranges to enforce specific configuration, or giving reference angles only to those joints with highly multimodal behavior, such as the elbow in the 5 DOF arm.

Are the developed algorithms flexible to be applicable to a wide range of problems and capable of being tailored to domain-specific characteristics simultaneously?

Based on the findings from the final experiments on problems from fluid mechanics and robotics, we conclude that the proposed algorithmic pipeline is, in principle, applicable to problems from different domains. If the bias matches the nature of the problem, which is best estimated by domain experts, the MPNN is generally adaptable to various problems. Furthermore, the unit-aware GP method can be tailored to diverse problems that involve quantities with units, which was demonstrated also beyond the realm of fluid mechanics and robotics. Nonetheless, it is important to note that certain restrictions exist that must be considered. Difficulties with multimodal problems became apparent in our experiments and must be addressed accordingly. This issue is not inherent to the proposed pipeline itself, but rather pertains to a common challenge faced by many learning algorithms, including MPNNs and GP. Within the two-step pipeline, errors introduced in the first stage can amplify in the second stage, which requires expert knowledge to estimate which error range is acceptable for the specific problem. Moreover, when additional domain knowledge is integrated into the algorithm, it must be designed carefully to avoid introducing additional complexities that could hinder the efficacy of the algorithm.

8.7 Summary

In this chapter, we applied a method for unit-aware GP that included constants

with undetermined units. Constants introduce "joker" units, which are propagated through the tree according to a propagation scheme. The dimensional analysis returns the magnitude of unit violations of an equation. Three constraint handling methods were assessed to minimize unit-violating individuals during evolution: evolutive culling, a repair mechanism, and a multi-objective approach. Experiments conducted on datasets of known equations showed that both evolutive culling and the multi-objective approach perform as well as a baseline method without dimensional analysis. The repair mechanism often introduced more constants than necessary, which was an undesired behavior. In-depth analysis of the PO fronts for benchmark datasets without ground truth revealed that a large share of solutions in the PO front of the baseline algorithm exhibited unit violations. All proposed unit-aware algorithms were able to identify solutions with similarly low error but without unit violations. However, evolutive culling and the repair mechanism tended to use more constants compared to the baseline algorithm.

Additionally, we applied the pipeline of proposed algorithmic advances to the two problems considered in this thesis. The resulting equations predicting the fluctuations around the mean drag exerted on a particle in particle-laden flows demonstrated accuracy values comparable to those of MPNNs, while remaining interpretable and consistent with known physical laws. For the IK problem in robotics, the integration of MPNNs as an inductive bias for GP produced mixed results. While the MPNN captured the complexity of the problem well for 3 DOF, the subsequent GP algorithm produced equations that mainly rely on the reference angle, which introduced strong local optima, and failed to learn the underlying physics. For 5 DOF, both the MPNN and GP had difficulties escaping the local optimum around the reference angle. Overall, these findings underline the importance of carefully designing problem representations and incorporating domain knowledge to avoid misleading the algorithm. Despite these limitations, the proposed ML pipeline for symbolic regression showed flexibility and adaptability, particularly in the fluid mechanics problem, and provides a strong foundation for further research.

9 Conclusion and Future Work

The following chapter concludes this thesis. We also provide an overview of topics for future research in the area of genetic programming for symbolic regression in applications from science and engineering.

9.1 Conclusion

In this thesis, we have proposed techniques to develop symbolic models for science and engineering applications using genetic programming and domain knowledge. We focused on graph-representable problems involving physical measurements, which are frequent characteristics of problems in these domains. Two benchmark problems without ground truth solutions from fluid mechanics and robotics were proposed. We then applied state-of-the-art GP approaches that preceded the advancements presented in this thesis to both problems to establish comparative baselines. We assessed the interplay between multi-objective GP and island models to improve the repetition stability of GP algorithms, which is an important requirement in practice. In addition, we explored the potential of graph neural networks as an inductive bias for GP and used different methods to gain insights into the underlying computations. We also proposed multiple constraint handling techniques for unit-aware GP using an enhanced dimensional analysis method, and have analyzed their differences on various quality criteria. Finally, we evaluated the proposed algorithms on both problems, assessing their applicability across different application domains while verifying their ability to be tailored to the specific characteristics of each problem. In the following, we highlight the contributions of this thesis by revisiting the research questions asked in Chapter 1.

RQ 1: Which techniques exist to develop symbolic models for problems from science and engineering?

- RQ 1.1: What are the current developments in the area of physics-informed machine learning?
- RQ 1.2: What are the current challenges in GP for application to symbolic regression problems, and how are they addressed?
- RQ 1.3: How is domain knowledge integrated into state-of-the-art GP techniques, and how can these techniques be classified?

In Chapter 3, we examined the related work covering several techniques for symbolic regression in science and engineering applications. We pointed out that

in contemporary literature, an increased interest in the integration of domain knowledge into data-driven modeling can be observed, which can be considered a counter-development to earlier approaches that focused on developing algorithms independent of specific problems or human input. This resulted in the development of numerous recent methods in the area of physics-informed machine learning, both ANN-based and SR-based, often tailored to specific problem domains. We reviewed the literature on key components of the GP algorithm that are critical for developing meaningful symbolic models in practical applications, including constant fitting, distributed algorithms, and GP for high-dimensional problems. Despite considerable advancements over the last decades, we identified limitations and gaps in the literature. For example, most studies on island models for GP focus on single-objective optimization, largely overlooking the potential effects of multi-objective optimization. A few studies employ GNNs as an inductive bias for GP to tackle high-dimensional problems, which seems meaningful for problems from science and engineering that can often be naturally represented as graphs. In addition, we presented an overview and classification of numerous methods from the literature that incorporate domain knowledge into GP algorithms by introducing problem-specific biases in Tab. 3.2. Dimensionally aware and shape-constrained GP were identified as the two predominant trends in the literature, each offering a range of interesting and diverse approaches.

RQ 2: How can symbolic models for the inverse kinematics of arbitrary robotic manipulators be developed?

- RQ 2.1: What are suitable objective functions to achieve physically meaningful equations?
- RQ 2.2: How do different types of domain knowledge integrated into the algorithm influence its performance?
- RQ 2.3: What are the limitations of this approach?

In Chapter 4, we introduced the inverse kinematics problem from the area of robotics as a benchmark problem for the algorithms presented in this thesis. To establish a comparative baseline, we applied state-of-the-art GP approaches combined with observational, inductive, and learning biases to develop symbolic models for a single 5 DOF manipulator. We assessed three objective functions, namely an error, a correlation, and a dimension penalty objective fitting for the problem to develop physically meaningful equations. The combination of error and correlation outperformed all other combinations of objectives, indicating that the inclusion of a dimension penalty can lead to reduced performance in terms of RMSE. As a type of inductive bias, we proposed a novel IK-CCGP algorithm to account for the multimodality inherent to IK, and represent interactions within the kinematic chain algorithmically through cooperative coevolution. The consecutive learning of two consecutive joints produced a position RMSE in the magnitude of $3.4\text{cm} - 4.1\text{cm}$, whereas extending the process to three joints approximately doubled the error magnitude. We concluded from these experiments, that the IK-CCGP algorithm struggled to effectively determine the contributions of each joint within the kinematic chain, leading to local optima that were difficult to escape. Another limitation became apparent in the dimensional inconsistency of most equations, as well as the lack of fitting new constants within the equations. Methods to tackle these issues were presented in the further course of this thesis.

RQ 3: How can symbolic models describing particle-laden flows be developed?

- RQ 3.1: How can a GP algorithm benefit from building blocks provided by domain experts?
- RQ 3.2: How do the evolved symbolic models perform against state-of-the-art baseline methods?

RQ 3.3: What are the limitations of this approach?

In Chapter 5, we presented the problem of particle-laden flows in the Stokes regime from the fluid mechanics domain as another benchmark for our algorithms. We introduced expert knowledge by providing building blocks from the solution of the flow around a single particle in the function set of the GP algorithm. We assessed the performance of state-of-the-art algorithms combined with other types of domain knowledge on the prediction of the Stokes flow past two inline spherical particles with varying distances. The GP algorithm identified solutions that outperformed the superimposition method for all distances. This, however, was surpassed by a simple ANN with multiple orders of magnitude in terms of RMSE. A statistical analysis of the results led us to conclude that the primary limitation of the proposed approach is the increasing error observed as the distance between the two particles decreased. This impedes the applicability of this method for more complex particle arrangements, typically involving numerous particles. Consequently, these results suggested a shift in problem modeling focus towards predicting hydrodynamic forces directly, rather than continuous velocity fields. These results helped to gain insights into potential pitfalls when learning the Stokes flow velocity field, and were used as a comparative baseline and motivation for the subsequent advances presented.

RQ 4: Can the success rate of a GP algorithm be improved with island models?

RQ 4.1: How does the interplay between island models and multi-objective optimization influence the final results?

In initial GP results on the fluid mechanics problem, we observed a large spread in the performance when the algorithm was repeated multiple times. To tackle this issue, we investigated several configurations of multi-objective island models to enhance the success rate of GP algorithms in Chapter 6. The interplay between the objectives and IM configurations has not as yet been studied in detail. In our work, we explored various aspects of this algorithm, including different combinations of objectives, migration topologies between islands, the number of islands, and the distribution of migrations over the algorithm runtime. Since these factors span a large space of potential hyperparameters, we focused on a limited set of configurations that had been previously employed in related studies. In our experiments on two problems from fluid mechanics, we observed a predominant influence of the objective functions on the success rate of the algorithm. For the overall best-performing objectives, the results improved even further when an island model instead of a single population approach was used. In these cases, this happened independently of the exact IM configuration. We concluded from this study that the choice of objective functions has a larger impact on the final result than the exact IM configuration. Based on these results, we decided to employ the combination of error, correlation, and dimension penalty objectives, together with an IM configuration, which, moreover, involved no additional computational cost. Since the experiments were conducted on a limited set of parameters, we do not claim that these configurations always yield the best results. The study of other parameter settings, such as varying mutation and crossover rates or different terminal sets per island, deserve further attention in the future. Promising extensions to this study, moreover, include the application to more complex benchmark problems from different problem domains. While we focused on the success rate of the final results, it would be interesting to gain further insights into the interplay between IM and objectives by comparing the convergence behavior between single population and IM algorithms.

RQ 5: What is the potential of graph neural networks as an inductive bias for GP to discover unknown symbolic models for high-dimensional problems that can be modeled as graphs?

RQ 5.1: Can particle-laden flows be approximated with graph neural networks?

RQ 5.2: How do the resulting equations perform in terms of error and interpretability?

RQ 5.3: Can the inverse kinematics problem be learned with graph neural networks?

Both benchmark problems analyzed in this thesis provided motivation for using MPNNs as an inductive bias for GP. First, approximating pairwise interactions between particles with MPNNs naturally fits the prediction of hydrodynamic forces on a particle in the Stokes flow through an arrangement of multiple particles. It reduced the dimensionality of the problem and overcame the shortcomings of the baseline algorithm, which showed limited applicability to more complex particle arrangements beyond two particles. Scaling up to 30 particles, we assessed two underlying functional forms. The MPNN achieved accuracy values similar to those of state-of-the-art approaches, which confirms the general applicability of this method. We developed symbolic models to replace the network blocks, of which the nested function exhibited a higher usage of constants while maintaining similar functional forms compared to the simple sum. This increased model complexity did not result in a significant decrease in error, which is why we concluded from these experiments that a simple sum over the pairwise interactions was most appropriate for this problem. The equations consistently performed slightly worse than the MPNN, which was, however, compensated by their simplicity and repeatedly occurring building blocks that helped to gain insights into the underlying computations. This method advanced the state-of-the-art approaches by introducing simple symbolic models for Stokes flow, a milestone not previously accomplished.

The baseline algorithm for the IK problem using cooperative coevolution to approximate interactions between manipulator joints faced local optima, reducing the accuracy and trustworthiness of the final solutions. MPNNs with their ability to pass messages between entities were considered a well-suited inductive bias for enabling information exchange between joints within the kinematic chain and enhancing collaboration to reach a final pose. Using a reference angle-guided approach, MPNNs were tasked with learning the IK of 3 and 5 DOF manipulators. The mean position error of approximately 1cm for the 3 DOF manipulator was reasonable for this type, although 4cm for the 5 DOF manipulator may still be acceptable depending on the specific task. The simpler 3 DOF manipulator was easier to approximate, as implied by the analysis of feature importance values. We concluded from this analysis that the MPNN for the 3 DOF manipulator successfully learned the underlying physics, while the 5 DOF manipulator mainly relied on the reference angle and, thus, requires further investigation. A more detailed discussion on the performance of MPNNs on both problems was provided in Sec. 7.3.

RQ 6: How can unit-aware equations be developed that include free constants?

RQ 6.1: How can undetermined units of free constants be considered during dimensional analysis?

RQ 6.2: What are suitable methods to handle unit constraints?

A major gap in the recent literature, which also became apparent in the baseline experiments in this thesis on both problems, is the ability to learn new constants in symbolic models and develop unit-aware equations at the same time. New constants come with unknown units, which cannot be propagated through the GP tree using traditional dimensional analysis. In Chapter 8, we employed an adapted dimensional analysis that propagated the units of constants as jokers through the tree and returned a magnitude of dimension violations within the tree. We proposed three constraint handling techniques to

consider unit constraints during evolution, namely evolutive culling, a repair mechanism, and a multi-objective approach, and assessed them on datasets with known and unknown ground truth. Among these techniques, evolutive culling and the multi-objective approach were able to identify the ground truth models correctly, even in the presence of increased noise in the data. From the analysis of PO fronts on datasets without ground truth, we concluded that the multi-objective approach produces more densely covered PO fronts with both unit-conformal and unit-violating solutions at lower complexities. Conversely, the other methods possessed gaps in the PO fronts where no unit-conformal solutions were identified. This led us to conclude that the multi-objective approach was most suited for the problems studied, offering unit-violating alternatives to the decision makers. Methods to further confirm these findings include the computations of crowding distance values of the final PO front.

Although discussed in detail in Sec. 8.6, we would like to conclude the results on the final research question below.

RQ 7: Are the developed algorithms flexible to be applicable to a wide range of problems and capable of being tailored to domain-specific characteristics simultaneously?

RQ 7.1: How do the algorithmic advances presented in this thesis perform in predicting particle-laden flows and inverse kinematics, compared to the baseline methods?

RQ 7.2: What are the limitations of these methods?

In Sec. 8.4 and 8.5, we applied the complete pipeline of algorithmic advances presented in this thesis — island models, MPNN as an inductive bias for GP, and a unit-aware GP algorithm with a dimension penalty as an additional objective — combined with additional expert-defined biases to both problems from fluid mechanics and robotics. For fluid mechanics problems, the proposed ML pipeline yielded improved results, almost closing the gap between ANN-based and GP predictions compared to the baseline method. We were able to provide meaningful building blocks and insights into the underlying computations to the domain experts. By contrast, we obtained mixed results for the inverse kinematics problem. The MPNN performed well for the 3 DOF manipulator, while for the 5-DOF manipulator, the network primarily relied on the reference angle. The equations replacing the network blocks largely relied on the reference angle provided in the dataset. We concluded from these results that the learning power of GP was limited in overcoming the local optimum introduced by the reference angle, which was introduced to the dataset as an observational bias. The MPNN faced similar problems for the 5 DOF manipulators but demonstrated enough learning capacity to infer the IK for the 3 DOF manipulator without heavily relying on the reference angles. The limitations we observed in our experiments were most obvious in the IK experiments, where the multimodality inherent to the problem, together with the reference angle introduced to tackle this issue, led to suboptimal results. This behavior, however, cannot be considered a failure of the proposed ML pipeline itself, but indicates a requirement for better techniques to handle multimodal problems. Neither MPNNs nor GP in its canonical form are designed to handle multimodality, which opens space for integrating more suitable biases from domain experts to enhance the results. Despite these challenges, we conclude that these experiments demonstrate the adaptability and generalizability of the proposed approach, provided that the chosen biases are well-suited to the problem at hand. The results moreover emphasize the importance of carefully designing biases to ensure they truly benefit the algorithm, which requires both a profound understanding of the problem and insight into the properties of the applied algorithms.

Throughout this thesis, we established comparative baselines for two problems which have an unknown ground truth and have not been studied in terms of

symbolic regression before, which can also be used in future research. We demonstrated a step-wise integration of domain knowledge to overcome shortcomings of the baseline and tackle increasing problem complexities. We are the first to have applied this pipeline to both problems, and consequently have advanced the state-of-the-art to fluid mechanics with our symbolic models. Despite the mixed results on the robotics problem, the pipeline has the potential to achieve higher accuracy with an improved handling of multimodality, which is the task of domain experts. A potential weak spot of this thesis is that the used GP software implementations together with their standard settings changed over time, which complicated the comparability between the baseline and advanced approaches. This is because the availability of GP frameworks that allow for the implementation of custom algorithm components, and fulfill important practical requirements at the same time, such as constant fitting or island model support, is limited. Moreover, the modeling of the problem changed over time due to different types of biases inserted to handle increasing problem complexities. As a result, the comparison of the fluid mechanics problems was mainly possible in a qualitative manner, such as how much better/worse did the equations perform compared to an ANN-based approach. Another aspect that was only covered partially was the selection procedure of equations that impact the evaluation of algorithms. In most cases, and when time was available, the selection of solutions was conducted by domain experts. In other cases, selection was performed with an automated approach from the final PO front using different quality criteria, such as the top $x\%$ method in Secs. 8.4 and 8.5. While we tried to make the selection process transparent for all experiments, for example, in Figs. 8.6 and 8.7, more sophisticated methods to reduce the number of solutions presented to decision makers are required. To this end, the following two approaches could provide a promising starting point: a cone-domination approach to identify knee points in the PO front, and an equality check to filter out solutions that differ in genotype but are identical in phenotype, such as $z + 1$ and $1 + z$, to give a simplified example.

Final Considerations on the Inclusion of Domain Knowledge for Practical Applications

To conclude, we demonstrated through various experiments how domain knowledge can enhance the equations evolved with GP, while also showing, in one instance, how it can lead to undesirable outcomes. This raises broader philosophical questions about how much domain knowledge should be integrated into our algorithms. While we cannot discuss this issue in its entirety, we know that too much bias can hinder the exploration of new models, while too little can result in unsatisfactory solutions. Finding the right balance, i.e., simplifying the problem enough to make it manageable, but not so much that the simplified version becomes a local optimum, remains challenging, and often requires trial and error. Nonetheless, we have shown in many cases how biases can reduce the problem complexity for the algorithm and improve results. Many of these biases were at the observational level, suggesting that the availability of the right data in the right form — such as ensuring all features are in SI units, or transforming positions into spherical coordinates — can already provide notable simplifications for the algorithm, without changing the algorithm itself. Overall, we conclude that suitable domain knowledge and close collaboration with domain experts is an essential step towards identifying meaningful symbolic models.

9.2 Future Work

Every journey eventually reaches its end, and so does this thesis. While we have made progress in addressing key aspects of genetic programming for symbolic regression in science and engineering, we do not claim to have answered all relevant questions. Many interesting ideas for future research arise from the results of this thesis, and we would like to specifically highlight the following five:

- Concerning the two benchmark problems proposed in this thesis, various extensions to the conducted experiments are possible. Regarding the problem of particle-laden flows, we developed models to predict the deviations from the mean drag on a particle in the Stokes regime, which is surrounded by numerous other particles. First, we can apply the proposed ML pipeline to the other components acting on a particle, namely lift and torque. Second, the promising results for the Stokes flow open space for further exploration of the capabilities of the algorithm when applied to higher Reynolds numbers, which is typically considered a more complex problem since flow patterns become asymmetric. As discussed in detail in several places, the measures to overcome the multimodality of the IK problem did not lead to satisfactory results. Therefore, applying the same algorithms with an improved method to handle multimodality seems promising. One possible approach would be to filter the dataset to include only a single elbow configuration, which could be useful for applications where only one configuration is allowed, such as when the elbow must always remain down when operating under a table.

- Beyond the application areas of this thesis, our approaches demonstrate strong potential for application to other real-world problems in science and engineering, particularly those where domain knowledge is available and which can be naturally expressed as graphs. These include the modeling of swarm behaviors and interactions, so as to navigate swarms of drones through complex environments; traffic and transformation networks, for which symbolic models can give important insights on relevant dependencies for traffic management; and particle physics, where symbolic models can discover underlying particle interactions. The proposed unit-aware GP algorithm including unknown constants can, of course, also be applied without the preceding step of MPNNs as an inductive bias. The recursive tree traversal algorithm for the adapted dimensional analysis is relatively simple, and contemporary GP frameworks already have multi-objective optimization implemented. Thus, we propose that our multi-objective unit-aware GP algorithm should be available for state-of-the-art frameworks in the future, to offer the possibility for physically meaningful and interpretable symbolic models.

- Although the code for most state-of-the-art GP frameworks is publicly available, we frequently encountered the challenge of their limited capacity to integrate domain knowledge throughout the course of this thesis. Some frameworks were tuned for performance in a way that complicates the integration of new code due to many dependencies. Others lacked the possibility to fit new constants, which is an indispensable requirement in contemporary symbolic regression, or did not support critical features, such as control of function nesting or custom complexity values for functions. Of course, it is not a trivial task to develop and maintain frameworks that are flexible enough to integrate new code ideas, while at the same time capable of being optimized for a reasonable computational efficiency. We found the `TiSR` framework [170] to balance these conflicting requirements well, but in the future it would be desirable to have more freedom to integrate domain knowledge into existing algorithms.

  Based on the experience gained in this thesis, certain framework properties can contribute to the integration of domain knowledge. Domain knowledge on the observational level is mainly related to pre-processing steps of data and not necessarily the task of the SR algorithm. Yet, software support for the conversion of features to SI units and for transformation from Cartesian to spherical coordinates would be desirable built-in features of future frameworks, as they are applicable to a wide range of problems. Domain knowledge on the inductive and learning levels has a higher influence on the algorithm itself, requiring a modular code structure where subcomponents can be easily replaced by others. On the

inductive level, custom function sets beyond arithmetic operations are a key feature. Custom evaluation functions and a selection of constraint handling methods for expert-defined constraints are probably the most important features to enable the integration of domain knowledge on the learning level.

- The final PO front can contain a large number of solutions, some of which are almost identical or only slightly different. A frequently encountered challenge throughout this thesis was the selection of symbolic models from the final PO front, which can be time-consuming for domain experts. Making trade-offs between accuracy and complexity, which we introduced as selection bias in Sec. 3.3, can be linked to the research area of decision-making. While identifying knee points of the PO front is one potential idea, more advanced methods might be required to handle the full complexity of the selection process and consider other requirements such as interpretability, robustness, and domain-specific constraints that were not optimized by the algorithm. Future research should consider decision-making methods and be targeted towards reducing the number of solutions presented to the decision maker to one or only a few. To this end, studies based on human feedback to develop heuristics for expert preferences to select from fronts could be a good starting point.

- A characteristic shared by many problems from science and engineering, including the two use cases studied in this thesis, is the availability of categorical data. Consequently, multiple parameter configurations of the otherwise identical system, such as varying volume fractions in particle laden flows, or different link lengths of robotic manipulators, are possible. While preliminary experiments beyond the scope of this thesis indicated that MPNN-based approaches yielded good results when trained on data with multiple categories simultaneously, the GP algorithms faced difficulties in incorporating the categorical variables and identifying accurate equations. For such cases, it is desirable to identify symbolic models that share the same structure but have different parameter values for each category. The multiview SR approach proposed in [236] can be considered a promising starting point to this end. Future research in this area includes the possibility for categorical and non-categorical constants, of which the latter would take on the same value for all categories. Given how frequently categorical data is available in experiments from science and engineering, applying categorical GP together with our unit-aware approach to such datasets opens a vast space of new application areas.

Bibliography

[1] Aggarwal, C. C. *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2023. DOI: 10.1007/978-3-031-29642-0.

[2] Akiki, G., Jackson, T. L., and Balachandar, S. "Pairwise interaction extended point-particle model for a random array of monodisperse spheres". In: *Journal of Fluid Mechanics* 813 (Feb. 2017), pp. 882–928. DOI: 10.1017/jfm.2016.877.

[3] Akiki, G., Moore, W. C., and Balachandar, S. "Pairwise-interaction extended point-particle model for particle-laden flows". In: *Journal of Computational Physics* 351 (Dec. 2017), pp. 329–357. DOI: 10.1016/j.jcp.2017.07.056.

[4] Aldeia, G. S. I. and França, F. O. de. "Lightweight Symbolic Regression with the Interaction - Transformation Representation". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. Rio de Janeiro: IEEE, July 2018, pp. 1–8. DOI: 10.1109/CEC.2018.8477951.

[5] Almusawi, A. R. J., Dülger, L. C., and Kapucu, S. "A New Artificial Neural Network Approach in Solving Inverse Kinematics of Robotic Arm (Denso VP6242)". In: *Computational Intelligence and Neuroscience* 2016 (2016), pp. 1–10. DOI: 10.1155/2016/5720163.

[6] Alpaydın, E. *Introduction to machine learning*. Fourth edition. Adaptive computation and machine learning. Cambridge, Massachusetts London: The MIT Press, 2020.

[7] Andre, D. and Koza, J. R. "A parallel implementation of genetic programming that achieves super-linear performance". In: *Information Sciences* 106.3-4 (May 1998), pp. 201–218. DOI: 10.1016/S0020-0255(97)10011-1.

[8] Angeline, P. J. "Comparing subtree crossover with macromutation". In: *Evolutionary Programming VI*. Ed. by Goos, G., Hartmanis, J., Van Leeuwen, J., Angeline, P. J., Reynolds, R. G., McDonnell, J. R., and Eberhart, R. Vol. 1213. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 101–111.

[9] Aristidou, A., Lasenby, J., Chrysanthou, Y., and Shamir, A. "Inverse Kinematics Techniques in Computer Graphics: A Survey". In: *Computer Graphics Forum* 37.6 (Sept. 2018), pp. 35–58. DOI: 10.1111/cgf.13310.

[10] Arnaldo, I., Krawiec, K., and O'Reilly, U.-M. "Multiple regression genetic programming". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. Vancouver BC Canada: ACM, July 2014, pp. 879–886. DOI: 10.1145/2576768.2598291.

[11] Arslan, S. and Ozturk, C. "Multi Hive Artificial Bee Colony Programming for high dimensional symbolic regression with feature selection". In: *Applied Soft Computing* 78 (May 2019), pp. 515–527. DOI: 10.1016/j.asoc.2019.03.014.

[12] Asadzadeh, M. Z., Gänser, H.-P., and Mücke, M. "Symbolic regression based hybrid semiparametric modelling of processes: An example case of a bending process". In: *Applications in Engineering Science* 6 (June 2021), p. 100049. DOI: 10.1016/j.apples.2021.100049.

[13] Baker, N., Alexander, F., Bremer, T., Hagberg, A., Kevrekidis, Y., Najm, H., Parashar, M., Patra, A., Sethian, J., Wild, S., Willcox, K., and Lee, S. *Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence*. Tech. rep. None, 1478744. Feb. 2019, None, 1478744. DOI: 10.2172/1478744.

[14] Bakurov, I., Buzzelli, M., Castelli, M., Vanneschi, L., and Schettini, R. "General Purpose Optimization Library (GPOL): A Flexible and Efficient Multi-Purpose Optimization Library in Python". In: *Applied Sciences* 11.11 (May 2021), p. 4774. DOI: 10.3390/app11114774.

[15] Balachandar, S. and Eaton, J. K. "Turbulent Dispersed Multiphase Flow". In: *Annual Review of Fluid Mechanics* 42.1 (Jan. 2010), pp. 111–133. DOI: 10.1146/annurev.fluid.010908.165243.

[16] Balachandar, S., Moore, W. C., Akiki, G., and Liu, K. "Toward particle-resolved accuracy in Euler–Lagrange simulations of multiphase flow using machine learning and pairwise interaction extended point-particle (PIEP) approximation". In: *Theoretical and Computational Fluid Dynamics* 34.4 (Aug. 2020), pp. 401–428. DOI: 10.1007/s00162-020-00538-8.

[17] Bandaru, S. and Deb, K. "A Dimensionally-Aware Genetic Programming Architecture for Automated Innovization". In: *Evolutionary Multi-Criterion Optimization*. Ed. by Purshouse, R. C., Fleming, P. J., Fonseca, C. M., Greco, S., and Shaw, J. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 513–527. DOI: 10.1007/978-3-642-37140-0_39.

[18] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. *Relational inductive biases, deep learning, and graph networks*. Oct. 2018. URL: http://arxiv.org/abs/1806.01261.

[19] Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D., and Kavukcuoglu, K. *Interaction Networks for Learning about Objects, Relations and Physics*. Dec. 2016. URL: http://arxiv.org/abs/1612.00222.

[20] Beyer, H.-G. and Schwefel, H.-P. "Evolution strategies – A comprehensive introduction". In: *Natural Computing* 1.1 (2002), pp. 3–52. DOI: 10.1023/A:1015059928466.

[21] Bihlo, A. "Improving physics-informed neural networks with meta-learned optimization". In: *Journal of Machine Learning Research* 25.14 (2024), pp. 1–26.

[22] Bishop, C. M. *Pattern recognition and machine learning*. Information science and statistics. New York: Springer, 2006.

[23] Björck, Å. *Numerical methods for least squares problems*. Philadelphia, Pa.: Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1996.

[24] Błądek, I. and Krawiec, K. "Solving symbolic regression problems with formal constraints". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Prague Czech Republic: ACM, July 2019, pp. 977–984. DOI: 10.1145/3321707.3321743.

[25] Boisbunon, A., Fanara, C., Grenet, I., Daeden, J., Vighi, A., and Schoenauer, M. "Zoetrope Genetic Programming for Regression". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. June 2021, pp. 776–784. DOI: 10.1145/3449639.3459349.

[26] Bomarito, G., Townsend, T., Stewart, K., Esham, K., Emery, J., and Hochhalter, J. "Development of interpretable, data-driven plasticity models with symbolic regression". In: *Computers & Structures* 252 (Aug. 2021), p. 106557. DOI: 10.1016/j.compstruc.2021.106557.

[27] Box, G. E. P. and Draper, N. R. *Empirical model-building and response surfaces*. Wiley series in probability and mathematical statistics. New York: Wiley, 1987.

[28] Brameier, M. and Banzhaf, W. "A comparison of linear genetic programming and neural networks in medical data mining". In: *IEEE Transactions on Evolutionary Computation* 5.1 (Feb. 2001), pp. 17–26. DOI: 10.1109/4235.910462.

[29] Broløs, K. R., Machado, M. V., Cave, C., Kasak, J., Stentoft-Hansen, V., Batanero, V. G., Jelen, T., and Wilstrup, C. *An Approach to Symbolic Regression Using Feyn*. Apr. 2021. URL: http://arxiv.org/abs/2104.05417.

[30] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (July 2017), pp. 18–42. DOI: 10.1109/MSP.2017.2693418.

[31] Brunton, S. L. and Kutz, J. N. *Machine Learning for Partial Differential Equations*. Mar. 2023. URL: http://arxiv.org/abs/2303.17078.

[32] Brunton, S. L., Proctor, J. L., and Kutz, J. N. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (Apr. 2016), pp. 3932–3937. DOI: 10.1073/pnas.1517384113.

[33] Buckingham, E. "On Physically Similar Systems; Illustrations of the Use of Dimensional Equations". In: *Physical Review* 4.4 (Oct. 1914), pp. 345–376. DOI: 10.1103/PhysRev.4.345.

[34] Burlacu, B., Kronberger, G., and Kommenda, M. "Operon C++: an efficient genetic programming framework for symbolic regression". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. Cancún Mexico: ACM, July 2020, pp. 1562–1570. DOI: 10.1145/3377929.3398099.

[35] Burlacu, B., Yang, K., and Affenzeller, M. "Population diversity and inheritance in genetic programming for symbolic regression". In: *Natural Computing* (Jan. 2023). DOI: 10.1007/s11047-022-09934-x.

[36] Capecelatro, J. and Desjardins, O. "An Euler–Lagrange strategy for simulating particle-laden flows". In: *Journal of Computational Physics* 238 (Apr. 2013), pp. 1–31. DOI: 10.1016/j.jcp.2012.12.015.

[37] Castelli, M., Manzoni, L., Silva, S., and Vanneschi, L. "A comparison of the generalization ability of different genetic programming frameworks". In: *IEEE Congress on Evolutionary Computation*. Barcelona, Spain: IEEE, July 2010, pp. 1–8. DOI: 10.1109/CEC.2010.5585925.

[38] Castelli, M., Vanneschi, L., Manzoni, L., and Popovič, A. "Semantic genetic programming for fast and accurate data knowledge discovery". In: *Swarm and Evolutionary Computation* 26 (Feb. 2016), pp. 1–7. DOI: 10.1016/j.swevo.2015.07.001.

[39] Chapelle, F. and Bidaud, P. "A closed form for inverse kinematics approximation of general 6R manipulators using genetic programming". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*. Vol. 4. Seoul, South Korea: IEEE, 2001, pp. 3364–3369. DOI: 10.1109/ROBOT.2001.933137.

[40] Chapelle, F. and Bidaud, P. "Closed form solutions for inverse kinematics approximation of general 6R manipulators". In: *Mechanism and Machine Theory* 39.3 (Mar. 2004), pp. 323–338. DOI: 10.1016/j.mechmachtheory.2003.09.003.

[41] Chapelle, O., Schölkopf, B., and Zien, A., eds. *Semi-supervised learning*. Adaptive computation and machine learning series. Cambridge, Mass.: MIT Press, 2010.

[42] Chen, C., Luo, C., and Jiang, Z. "A multilevel block building algorithm for fast modeling generalized separable systems". In: *Expert Systems with Applications* 109 (Nov. 2018), pp. 25–34. DOI: 10.1016/j.eswa.2018.05.021.

[43] Chen, C., Luo, C., and Jiang, Z. "Block building programming for symbolic regression". In: *Neurocomputing* 275 (Jan. 2018), pp. 1973–1980. DOI: 10.1016/j.neucom.2017.10.047.

[44] Chen, Q. "Improving the Generalisation of Genetic Programming for Symbolic Regression". PhD thesis. Te Herenga Waka - Victoria University of Wellington, 2018.

[45] Chen, Q., Xue, B., Banzhaf, W., and Zhang, M. "Relieving Genetic Programming from Coefficient Learning for Symbolic Regression via Correlation and Linear Scaling". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Lisbon Portugal: ACM, July 2023, pp. 420–428. DOI: 10.1145/3583131.3595918.

[46] Chen, Q., Xue, B., Niu, B., and Zhang, M. "Improving generalisation of genetic programming for high-dimensional symbolic regression with feature selection". In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. Vancouver, BC, Canada: IEEE, July 2016, pp. 3793–3800. DOI: 10.1109/CEC.2016.7744270.

[47] Chen, Q., Xue, B., and Zhang, M. "Generalisation and domain adaptation in GP with gradient descent for symbolic regression". In: *2015 IEEE Congress on Evolutionary Computation (CEC)*. Sendai, Japan: IEEE, May 2015, pp. 1137–1144. DOI: 10.1109/CEC.2015.7257017.

[48] Chen, Q., Xue, B., and Zhang, M. "Improving Generalization of Genetic Programming for Symbolic Regression With Angle-Driven Geometric Semantic Operators". In: *IEEE Transactions on Evolutionary Computation* 23.3 (June 2019), pp. 488–502. DOI: 10.1109/TEVC.2018.2869621.

[49] Chen, Q., Xue, B., and Zhang, M. "Rademacher Complexity for Enhancing the Generalization of Genetic Programming for Symbolic Regression". In: *IEEE Transactions on Cybernetics* 52.4 (Apr. 2022), pp. 2382–2395. DOI: 10.1109/TCYB.2020.3004361.

[50] Chen, Q., Zhang, M., and Xue, B. "Feature Selection to Improve Generalization of Genetic Programming for High-Dimensional Symbolic Regression". In: *IEEE Transactions on Evolutionary Computation* 21.5 (Oct. 2017), pp. 792–806. DOI: 10.1109/TEVC.2017.2683489.

[51] Chen, Q., Zhang, M., and Xue, B. "Genetic Programming with Embedded Feature Construction for High-Dimensional Symbolic Regression". In: *Intelligent and Evolutionary Systems*. Ed. by Leu, G., Singh, H. K., and Elsayed, S. Vol. 8. Cham: Springer International Publishing, 2017, pp. 87–102. DOI: 10.1007/978-3-319-49049-6_7.

[52] Chen, Q., Zhang, M., and Xue, B. "Structural Risk Minimization-Driven Genetic Programming for Enhancing Generalization in Symbolic Regression". In: *IEEE Transactions on Evolutionary Computation* 23.4 (Aug. 2019), pp. 703–717. DOI: 10.1109/TEVC.2018.2881392.

[53] Cherrier, N., Poli, J.-P., Defurne, M., and Sabatie, F. "Consistent Feature Construction with Constrained Genetic Programming for Experimental Physics". In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. Wellington, New Zealand: IEEE, June 2019, pp. 1650–1658. DOI: 10.1109/CEC.2019.8789937.

[54] Cohen, B. G., Beykal, B., and Bollas, G. M. "Physics-informed genetic programming for discovery of partial differential equations from scarce and noisy data". In: *Journal of Computational Physics* 514 (Oct. 2024), p. 113261. DOI: 10.1016/j.jcp.2024.113261.

[55] Cornelio, C., Dash, S., Austel, V., Josephson, T. R., Goncalves, J., Clarkson, K. L., Megiddo, N., El Khadir, B., and Horesh, L. "Combining data and theory for derivable scientific discovery with AI-Descartes". In: *Nature Communications* 14.1 (Apr. 2023), p. 1777. DOI: 10.1038/s41467-023-37236-y.

[56] Cortez, R. "The Method of Regularized Stokeslets". In: *SIAM Journal on Scientific Computing* 23.4 (Jan. 2001), pp. 1204–1225. DOI: 10.1137/S106482750038146X.

[57] Cranmer, M. *Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl*. May 2023. URL: http://arxiv.org/abs/2305.01582.

[58] Cranmer, M., Ashok, D., Coxon, T., Booth-Clibborn, W., Harold, W. K., Brehmer, J., Kittisopikul, M., Aluthge, D., foxtran, Maheshkar, S., Mahdi Hasan, S., Grundner, A., BrotherHa, Bot, D., Tsoi, H. F., Wang, H., Orson, I., Wadekar, D., LionessOfCintra, Peralta Lozada, R., Hvaara, R., Mengel, T., Jelen, T., Jain, V., and Thompson, W. *MilesCranmer/PySR: v1.3.1*. Dec. 2024. URL: https://zenodo.org/doi/10.5281/zenodo.14560734.

[59] Cranmer, M., Greydanus, S., Hoyer, S., Battaglia, P., Spergel, D., and Ho, S. *Lagrangian Neural Networks*. July 2020. URL: http://arxiv.org/abs/2003.04630.

[60] Cranmer, M., Sanchez Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. "Discovering Symbolic Models from Deep Learning with Inductive Biases". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 17429–17442.

[61] Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., and Piccialli, F. "Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next". In: *Journal of Scientific Computing* 92.3 (Sept. 2022), p. 88. DOI: 10.1007/s10915-022-01939-z.

[62] d'Ascoli, S., Becker, S., Mathis, A., Schwaller, P., and Kilbertus, N. *ODEFormer: Symbolic Regression of Dynamical Systems with Transformers*. Oct. 2023. URL: http://arxiv.org/abs/2310.05573.

[63] Daunicht, W. "Approximation of the inverse kinematics of an industrial robot by DEFAnet". In: *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*. Singapore: IEEE, 1991, 1995–2000 vol.3. DOI: 10.1109/IJCNN.1991.170676.

[64] De Silva, B. M., Higdon, D. M., Brunton, S. L., and Kutz, J. N. "Discovery of Physics From Data: Universal Laws and Discrepancies". In: *Frontiers in Artificial Intelligence* 3 (Apr. 2020), p. 25. DOI: 10.3389/frai.2020.00025.

[65] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197. DOI: 10.1109/4235.996017.

[66] Deb, K. *Multi-objective optimization using evolutionary algorithms*. 1st ed. Wiley-Interscience series in systems and optimization. Chichester ; New York: John Wiley & Sons, 2001.

[67] Demby's, J., Gao, Y., and DeSouza, G. N. "A Study on Solving the Inverse Kinematics of Serial Robots using Artificial Neural Network and Fuzzy Neural Network". In: *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. New Orleans, LA, USA: IEEE, June 2019, pp. 1–6. DOI: 10.1109/FUZZ-IEEE.2019.8858872.

[68] Dick, G. "Bloat and Generalisation in Symbolic Regression". In: *Simulated Evolution and Learning*. Ed. by Dick, G., Browne, W. N., Whigham, P., Zhang, M., Bui, L. T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K. C., and Tang, K. Vol. 8886. Cham: Springer International Publishing, 2014, pp. 491–502. DOI: 10.1007/978-3-319-13563-2_42.

[69] Dick, G. "Sensitivity-like analysis for feature selection in genetic programming". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Berlin Germany: ACM, July 2017, pp. 401–408. DOI: 10.1145/3071178.3071338.

[70] Dick, G., Owen, C. A., and Whigham, P. A. "Feature standardisation and coefficient optimisation for effective symbolic regression". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. Cancún Mexico: ACM, June 2020, pp. 306–314. DOI: 10.1145/3377930.3390237.

[71] Dolinsky, J., Jenkinson, I., and Colquhoun, G. "Application of genetic programming to the calibration of industrial robots". In: *Computers in Industry* 58.3 (Apr. 2007), pp. 255–264. DOI: 10.1016/j.compind.2006.06.003.

[72] Dorigo, M. and Stützle, T. *Ant colony optimization*. A Bradford book. Cambridge, Mass.: MIT Press, 2004.

[73] Durasević, M., Jakobovic, D., Scoczynski Ribeiro Martins, M., Picek, S., and Wagner, M. "Fitness Landscape Analysis of Dimensionally-Aware Genetic Programming Featuring Feynman Equations". In: *Parallel Problem Solving from Nature – PPSN XVI*. Ed. by Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M., and Trautmann, H. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 111–124. DOI: 10.1007/978-3-030-58115-2_8.

[74] Durasević, M., Jakobović, D., and Knežević, K. "Adaptive scheduling on unrelated machines with genetic programming". In: *Applied Soft Computing* 48 (Nov. 2016), pp. 419–430. DOI: 10.1016/j.asoc.2016.07.025.

[75] Ebner, M. "On the search space of genetic programming and its relation to nature's search space". In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. Washington, DC, USA: IEEE, 1999, pp. 1357–1361. DOI: 10.1109/CEC.1999.782609.

[76] Eckmiller, Beckmann, Werntges, and Lades. "Neural kinematics net for a redundant robot arm". In: *International Joint Conference on Neural Networks*. Washington, DC, USA: IEEE, 1989, 333–339 vol.2. DOI: 10.1109/IJCNN.1989.118719.

[77] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. "Least angle regression". In: *The Annals of Statistics* 32.2 (Apr. 2004). DOI: 10.1214/009053604000000067.

[78] Ekárt, A. and Németh, S. Z. "Selection Based on the Pareto Nondomination Criterion for Controlling Code Growth in Genetic Programming". In: *Genetic Programming and Evolvable Machines*. Genetic Programming and Evolvable Machines 2.1 (2001), pp. 61–73. DOI: 10.1023/A:1010070616149.

[79] Elmestikawy, H., Reuter, J., Evrard, F., Mostaghim, S., and Van Wachem, B. "Deterministic drag modelling for spherical particles in Stokes regime using data-driven approaches". In: *International Journal of Multiphase Flow* 178 (Aug. 2024), p. 104880. DOI: 10.1016/j.ijmultiphaseflow.2024.104880.

[80] Espada, G., Ingelse, L., Canelas, P., Barbosa, P., and Fonseca, A. "Data Types as a More Ergonomic Frontend for Grammar-Guided Genetic Programming". In: *Proceedings of the 21st ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. Auckland New Zealand: ACM, Nov. 2022, pp. 86–94. DOI: 10.1145/3564719.3568697.

[81] Fernández, F., Tomassini, M., Punch, W. F., and Sánchez, J. M. "Experimental Study of Multipopulation Parallel Genetic Programming". In: *Genetic Programming*. Ed. by Goos, G., Hartmanis, J., Leeuwen, J. van, Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P., and Fogarty, T. C. Vol. 1802. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 283–293. DOI: 10.1007/978-3-540-46239-2_21.

[82] Fernández, F., Tomassini, M., and Vanneschi, L. "Studying the Influence of Communication Topology and Migration on Distributed Genetic Programming". In: *Genetic Programming*. Ed. by Goos, G., Hartmanis, J., Leeuwen, J. van, Miller, J., Tomassini, M., Lanzi, P. L., Ryan, C., Tettamanzi, A. G. B., and Langdon, W. B. Vol. 2038. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 51–63. DOI: 10.1007/3-540-45355-5_5.

[83] Fernández, F. F., Tomassini, M., and Vanneschi, L. "An Empirical Study of Multipopulation Genetic Programming". In: *Genetic Programming and Evolvable Machines* 4 (2003), pp. 21–51. DOI: https://doi.org/10.1023/A:1021873026259.

[84] Ferreira, C. "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems". In: 13.2 (2001), pp. 7–129.

[85] Fey, M. and Lenssen, J. E. *Fast Graph Representation Learning with PyTorch Geometric*. Apr. 2019. URL: http://arxiv.org/abs/1903.02428.

[86] Fillon, C. and Bartoli, A. "A Divide & Conquer Strategy for Improving Efficiency and Probability of Success in Genetic Programming". In: *Genetic Programming*. Ed. by Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Collet, P., Tomassini, M., Ebner, M., Gustafson, S., and Ekárt, A. Vol. 3905. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 13–23. DOI: 10.1007/11729976_2.

[87] Fitzgerald, J., Azad, R. M. A., and Ryan, C. "A bootstrapping approach to reduce over-fitting in genetic programming". In: *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. Amsterdam The Netherlands: ACM, July 2013, pp. 1113–1120. DOI: 10.1145/2464576.2482690.

[88] Fitzgerald, J. and Ryan, C. "On size, complexity and generalisation error in GP". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. Vancouver BC Canada: ACM, July 2014, pp. 903–910. DOI: 10.1145/2576768.2598346.

[89] Fogel, D. B., Bäck, T., and Michalewicz, Z. *Evolutionary computation*. Bristol: IOP, 2000.

[90] Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A. G., Parizeau, M., and Gagné, C. "DEAP: evolutionary algorithms made easy". In: *The Journal of Machine Learning Research* 13.1 (July 2012), pp. 2171–2175.

[91] Franca, F. O. de, Virgolin, M., Kommenda, M., Majumder, M. S., Cranmer, M., Espada, G., Ingelse, L., Fonseca, A., Landajuela, M., Petersen, B., Glatt, R., Mundhenk, N., Lee, C. S., Hochhalter, J. D., Randall, D. L., Kamienny, P., Zhang, H., Dick, G., Simon, A., Burlacu, B., Kasak, J., Machado, M., Wilstrup, C., and La Cava, W. G. *Interpretable Symbolic Regression for Data Science: Analysis of the 2022 Competition*. July 2023. URL: http://arxiv.org/abs/2304.01117.

[92] França, F. O. de. "A greedy search tree heuristic for symbolic regression". In: *Information Sciences* 442-443 (May 2018), pp. 18–32. DOI: 10.1016/j.ins.2018.02.040.

[93] França, F. O. de and Aldeia, G. S. I. "Interaction–Transformation Evolutionary Algorithm for Symbolic Regression". In: *Evolutionary Computation* 29.3 (Sept. 2021), pp. 367–390. DOI: 10.1162/evco_a_00285.

[94] Garbrecht, K., Birky, D., Lester, B., Emery, J., and Hochhalter, J. "Complementing a continuum thermodynamic approach to constitutive modeling with symbolic regression". In: *Journal of the Mechanics and Physics of Solids* 181 (Dec. 2023), p. 105472. DOI: 10.1016/j.jmps.2023.105472.

[95] Géron, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Second edition. Beijing [China] ; Sebastopol, CA: O'Reilly Media, Inc, 2019.

[96] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. 2017, pp. 1263–1272.

[97] Goldberg, D. E. *Genetic algorithms in search, optimization, and machine learning*. Reading, Mass: Addison-Wesley Pub. Co, 1989.

[98] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016.

[99] Gori, M., Monfardini, G., and Scarselli, F. "A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005*. Vol. 2. Montreal, Que., Canada: IEEE, 2005, pp. 729–734. DOI: 10.1109/IJCNN.2005.1555942.

[100] Greydanus, S., Dzamba, M., and Yosinski, J. *Hamiltonian Neural Networks*. Sept. 2019. URL: http://arxiv.org/abs/1906.01563.

[101] Grundner, A., Beucler, T., Gentine, P., and Eyring, V. *Data-Driven Equation Discovery of a Cloud Cover Parameterization*. Feb. 2024. URL: http://arxiv.org/abs/2304.08063.

[102] Guazzelli, É., Morris, J. F., and Pic, S. *A physical introduction to suspension dynamics*. Cambridge texts in applied mathematics. Cambridge: Cambridge University Press, 2011.

[103] Gupta, M., Bahri, D., Cotter, A., and Canini, K. "Diminishing returns shape constraints for interpretability and regularization". In: *Advances in neural information processing systems*. Ed. by Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. Vol. 31. Curran Associates, Inc., 2018.

[104] Haider, C., De Franca, F., Burlacu, B., and Kronberger, G. "Shape-constrained multi-objective genetic programming for symbolic regression". In: *Applied Soft Computing* 132 (2022), p. 109855. DOI: 10.1016/j.asoc.2022.109855.

[105] Haider, C., França, F. O. de, Burlacu, B., Bachinger, F., Kronberger, G., and Affenzeller, M. "Shape-constrained Symbolic Regression: Real-World Applications in Magnetization, Extrusion and Data Validation". In: *Genetic Programming Theory and Practice XX*. Ed. by Winkler, S., Trujillo, L., Ofria, C., and Hu, T. Singapore: Springer Nature Singapore, 2024, pp. 225–240. DOI: 10.1007/978-981-99-8413-8_12.

[106] Haider, C., França, F. O. de, Kronberger, G., and Burlacu, B. "Comparing optimistic and pessimistic constraint evaluation in shape-constrained symbolic regression". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Boston Massachusetts: ACM, July 2022, pp. 938–945. DOI: 10.1145/3512290.3528714.

[107] Haider, C. and Kronberger, G. "Shape-Constrained Symbolic Regression with NSGA-III". In: *Computer Aided Systems Theory – EUROCAST 2022*. Ed. by Moreno-Díaz, R., Pichler, F., and Quesada-Arencibia, A. Vol. 13789. Cham: Springer Nature Switzerland, 2022, pp. 164–172. DOI: 10.1007/978-3-031-25312-6_19.

[108] He, B., Lu, Q., Yang, Q., Luo, J., and Wang, Z. "Taylor genetic programming for symbolic regression". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Boston Massachusetts: ACM, July 2022, pp. 946–954. DOI: 10.1145/3512290.3528757.

[109] He, Y., Aranha, C., and Sakurai, T. "Incorporating sub-programs as knowledge in program synthesis by PushGP and adaptive replacement mutation". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. Boston Massachusetts: ACM, July 2022, pp. 554–557. DOI: 10.1145/3520304.3528891.

[110] Al-Helali, B., Chen, Q., Xue, B., and Zhang, M. "Genetic Programming for Feature Selection Based on Feature Removal Impact in High-Dimensional Symbolic Regression". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 8.3 (June 2024), pp. 2269–2282. DOI: 10.1109/TETCI.2024.3369407.

[111] Hemberg, E., Kelly, J., and O'Reilly, U.-M. "On domain knowledge and novelty to improve program synthesis performance with grammatical evolution". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Prague Czech Republic: ACM, July 2019, pp. 1039–1046. DOI: 10.1145/3321707.3321865.

[112] Hochreiter, S. and Schmidhuber, J. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[113] Holland, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 1992. DOI: 10.7551/mitpress/1090.001.0001.

[114] Holt, S., Qian, Z., and Schaar, M. van der. *Deep Generative Symbolic Regression*. Dec. 2023. URL: http://arxiv.org/abs/2401.00282.

[115] Hornby, G. S. "ALPS: the age-layered population structure for reducing the problem of premature convergence". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. Seattle Washington USA: ACM, July 2006, pp. 815–822. DOI: 10.1145/1143997.1144142.

[116] Hornby, G. S. "Steady-state ALPS for real-valued problems". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. Montreal Québec Canada: ACM, July 2009, pp. 795–802. DOI: 10.1145/1569901.1570011.

[117] Hornik, K., Stinchcombe, M., and White, H. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[118] Hu, J. and Goodman, E. "The hierarchical fair competition (HFC) model for parallel evolutionary algorithms". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*. Vol. 1. Honolulu, HI, USA: IEEE, 2002, pp. 49–54. DOI: 10.1109/CEC.2002.1006208.

[119] Hu, J., Goodman, E. D., Seo, K., and Pei, M. "Adaptive hierarchical fair competition (AHFC) model for parallel evolutionary algorithms". In: *Proceedings of the 4th annual conference on genetic and evolutionary computation*. 2002, pp. 772–779.

[120] James, G., Witten, D., Hastie, T., and Tibshirani, R. *An introduction to statistical learning: with applications in R*. Second edition. Springer texts in statistics. New York, NY: Springer, 2021. DOI: 10.1007/978-1-0716-1418-1.

[121] Jazar, R. N. *Theory of Applied Robotics: Kinematics, Dynamics, and Control (2nd Edition)*. Boston, MA: Springer US, 2010.

[122] Jonyer, I. and Himes, A. "Improving Modularity in Genetic Programming Using Graph-Based Data Mining". In: Melbourne Beach, FL, USA, 2006, pp. 556–561.

[123] Kalra, P., Mahapatra, P., and Aggarwal, D. "An evolutionary approach for solving the multimodal inverse kinematics problem of industrial robots". In: *Mechanism and Machine Theory* 41.10 (Oct. 2006), pp. 1213–1229. DOI: 10.1016/j.mechmachtheory.2005.11.005.

[124] Kamienny, P.-A., Lample, G., and Charton, F. "End-to-end Symbolic Regression with Transformers". In: *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*. Ed. by Koyejo, S. Advances in Neural Information Processing Systems 35. Curran Associates, Inc., 2023, pp. 10269–10281.

[125] Kaptanoglu, A. A., Silva, B. M. d., Fasel, U., Kaheman, K., Goldschmidt, A. J., Callaham, J., Delahunt, C. B., Nicolaou, Z. G., Champion, K., Loiseau, J.-C., Kutz, J. N., and Brunton, S. L. "PySINDy: A comprehensive Python package for robust sparse system identification". In: *Journal of Open Source Software* 7.69 (Jan. 2022), p. 3994. DOI: 10.21105/joss.03994.

[126] Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (May 2021), pp. 422–440. DOI: 10.1038/s42254-021-00314-5.

[127] Keijzer, M. "Improving Symbolic Regression with Interval Arithmetic and Linear Scaling". In: *Genetic Programming*. Ed. by Goos, G., Hartmanis, J., Van Leeuwen, J., Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., and Costa, E. Vol. 2610. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 70–82.

[128] Keijzer, M. and Babovic, V. "Dimensionally Aware Genetic Programming". In: *GECCO'99: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*. Vol. 2. 1999, pp. 1069–1076.

[129] Kennedy, J. and Eberhart, R. "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. Perth, WA, Australia: IEEE, 1995, pp. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

[130] Keren, L. S., Liberzon, A., and Lazebnik, T. "A computational framework for physics-informed symbolic regression with straightforward integration of domain knowledge". In: *Scientific Reports* 13.1 (Jan. 2023), p. 1249. DOI: 10.1038/s41598-023-28328-2.

[131] Kerrigan, D., Hullman, J., and Bertini, E. "A Survey of Domain Knowledge Elicitation in Applied Machine Learning". In: *Multimodal Technologies and Interaction* 5.12 (Nov. 2021), p. 73. DOI: 10.3390/mti5120073.

[132] Kim, J. T., Park, J., Choi, S., and Ha, S. *Learning Robot Structure and Motion Embeddings using Graph Neural Networks*. Sept. 2021. URL: http://arxiv.org/abs/2109.07543.

[133] Kinzett, D., Johnston, M., and Zhang, M. "How online simplification affects building blocks in genetic programming". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. Montreal Québec Canada: ACM, July 2009, pp. 979–986. DOI: 10.1145/1569901.1570035.

[134] Kommenda, M. "Local Optimization and Complexity Control for Symbolic Regression". PhD thesis. Johannes Kepler Universität Linz, 2018.

[135] Kommenda, M., Affenzeller, M., Kronberger, G., and Winkler, S. M. "Nonlinear Least Squares Optimization of Constants in Symbolic Regression". In: *Computer Aided Systems Theory - EUROCAST 2013*. Ed. by Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Moreno-Díaz, R., Pichler, F., and Quesada-Arencibia, A. Vol. 8111. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 420–427. DOI: 10.1007/978-3-642-53856-8_53.

[136] Kommenda, M., Beham, A., Affenzeller, M., and Kronberger, G. "Complexity Measures for Multi-objective Symbolic Regression". In: *Computer Aided Systems Theory – EUROCAST 2015*. Ed. by Moreno-Díaz, R., Pichler, F., and Quesada-Arencibia, A. Vol. 9520. Cham: Springer International Publishing, 2015, pp. 409–416. DOI: 10.1007/978-3-319-27340-2_51.

[137] Kommenda, M., Burlacu, B., Kronberger, G., and Affenzeller, M. "Parameter identification for symbolic regression using nonlinear least squares". In: *Genetic Programming and Evolvable Machines* 21.3 (Sept. 2020), pp. 471–501. DOI: 10.1007/s10710-019-09371-3.

[138] Kommenda, M., Kronberger, G., Winkler, S., Affenzeller, M., and Wagner, S. "Effects of constant optimization by nonlinear least squares minimization in symbolic regression". In: *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. Amsterdam The Netherlands: ACM, July 2013, pp. 1121–1128. DOI: 10.1145/2464576.2482691.

[139] Kondor, R., Son, H. T., Pan, H., Anderson, B., and Trivedi, S. *Covariant Compositional Networks For Learning Graphs*. Jan. 2018. URL: http://arxiv.org/abs/1801.02144.

[140] Koza, J. R. *Genetic programming. 1: On the programming of computers by means of natural selection*. Complex adaptive systems. Cambridge, Mass.: MIT Press, 1992.

[141] Koza, J. R., Andre, D., Bennett, F., and Keane, M. A. "Use of Automatically Defined Functions and Architecture-Altering Operations in Automated Circuit Synthesis with Genetic Programming". In: *Genetic Programming 1996*. Ed. by Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L. The MIT Press, July 1996, pp. 132–140. DOI: 10.7551/mitpress/3242.003.0019.

[142] Krauss, O., Mössenböck, H., and Affenzeller, M. "Towards Knowledge-guided Genetic Improvement". In: *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. Seoul Republic of Korea: ACM, June 2020, pp. 293–294. DOI: 10.1145/3387940.3392172.

[143] Krishnapriyan, A. S., Gholami, A., Zhe, S., Kirby, R. M., and Mahoney, M. W. *Characterizing possible failure modes in physics-informed neural networks*. Nov. 2021. URL: http://arxiv.org/abs/2109.01050.

[144] Kronberger, G., De Franca, F. O., Burlacu, B., Haider, C., and Kommenda, M. "Shape-Constrained Symbolic Regression—Improving Extrapolation with Prior Knowledge". In: *Evolutionary Computation* 30.1 (Mar. 2022), pp. 75–98. DOI: 10.1162/evco_a_00294.

[145] Kronberger, G. "Symbolic Regression for Knowledge Discovery - Bloat, Overfitting, and Variable Interaction Networks". PhD thesis. Linz: Johannes Kepler Universität Linz, 2010.

[146] Kronberger, G., Kommenda, M., Promberger, A., and Nickel, F. "Predicting friction system performance with symbolic regression and genetic programming with factor variables". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Kyoto Japan: ACM, July 2018, pp. 1278–1285. DOI: 10.1145/3205455.3205522.

[147] Kruse, R., Mostaghim, S., Borgelt, C., Braune, C., Steinbrecher, M., Klawonn, F., and Moewes, C. *Computational intelligence: a methodological introduction*. Third edition. Texts in computer science. Cham, Switzerland: Springer, 2022. DOI: 10.1007/978-3-030-42227-1.

[148] Kubalík, J., Derner, E., and Babuška, R. "Multi-objective symbolic regression for physics-aware dynamic modeling". In: *Expert Systems with Applications* 182 (Nov. 2021), p. 115210. DOI: 10.1016/j.eswa.2021.115210.

[149] Kubalík, J., Derner, E., and Babuška, R. "Symbolic regression driven by training data and prior knowledge". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. Cancún Mexico: ACM, June 2020, pp. 958–966. DOI: 10.1145/3377930.3390152.

[150] Kucuk, S. and Bingul, Z. "Inverse kinematics solutions for industrial robot manipulators with offset wrists". In: *Applied Mathematical Modelling* 38.7-8 (Apr. 2014), pp. 1983–1999. DOI: 10.1016/j.apm.2013.10.014.

[151] La Cava, W., Helmuth, T., Spector, L., and Moore, J. H. "A Probabilistic and Multi-Objective Analysis of Lexicase Selection and -Lexicase Selection". In: *Evolutionary Computation* 27.3 (Sept. 2019), pp. 377–402. DOI: 10.1162/evco_a_00224.

[152] La Cava, W., Orzechowski, P., Burlacu, B., França, F. O. de, Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. "Contemporary Symbolic Regression Methods and their Relative Performance". In: ed. by Vanschoren, J. and Yeung, S. 2021. DOI: 10.48550/arXiv.2107.14351.

[153] La Cava, W., Singh, T. R., Taggart, J., Suri, S., and Moore, J. H. *Learning concise representations for regression by evolving networks of trees*. Mar. 2019. URL: http://arxiv.org/abs/1807.00981.

[154] La Cava, W., Spector, L., and Danai, K. "Epsilon-Lexicase Selection for Regression". In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. Denver Colorado USA: ACM, July 2016, pp. 741–748. DOI: 10.1145/2908812.2908898.

[155] Labonne, M. *Hands-on graph neural networks using Python: practical techniques and architectures for building powerful graph and deep learning apps with PyTorch*. Birmingham, UK: Packt Publishing Ltd., 2023.

[156] Landajuela, M., Lee, C. S., Yang, J., Glatt, R., Santiago, C. P., Aravena, I., Mundhenk, T., Mulcahy, G., and Petersen, B. K. "A unified framework for deep symbolic regression". In: *Advances in neural information processing systems*. Ed. by Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. Vol. 35. Curran Associates, Inc., 2022, pp. 33985–33998.

[157] Landajuela, M., Petersen, B. K., Kim, S., Santiago, C. P., Glatt, R., Mundhenk, T. N., Pettit, J. F., and Faissol, D. M. "Discovering symbolic policies with deep reinforcement learning". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Meila, M. and Zhang, T. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 5979–5989.

[158] Langdon, W. B. and Nordin, J. P. "Seeding Genetic Programming Populations". In: *Genetic Programming*. Ed. by Goos, G., Hartmanis, J., Van Leeuwen, J., Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P., and Fogarty, T. C. Vol. 1802. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 304–315. DOI: 10.1007/978-3-540-46239-2_23.

[159] Langley, P. "BACON: A Production System That Discovers Empirical Laws". In: *International Joint Conference on Artificial Intelligence*. 1977.

[160] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[161] Leijnen, S. and Veen, F. V. "The Neural Network Zoo". In: *IS4SI 2019 Summit*. MDPI, May 2020, p. 9. DOI: 10.3390/proceedings2020047009.

[162] Lemos, P., Jeffrey, N., Cranmer, M., Ho, S., and Battaglia, P. "Rediscovering orbital mechanics with machine learning". In: *Machine Learning: Science and Technology* 4.4 (Dec. 2023), p. 045002. DOI: 10.1088/2632-2153/acfa63.

[163] Li, D. and Zhong, J. "Dimensionally Aware Multi-Objective Genetic Programming for Automatic Crowd Behavior Modeling". In: *ACM Transactions on Modeling and Computer Simulation* 30.3 (July 2020), 19:1–19:24. DOI: 10.1145/3391407.

[164] Li, Y., Li, W., Yu, L., Wu, M., Liu, J., Li, W., Hao, M., Wei, S., and Deng, Y. *MetaSymNet: A Dynamic Symbolic Regression Network Capable of Evolving into Arbitrary Formulations*. Nov. 2023. URL: http://arxiv.org/abs/2311.07326.

[165] Li, Z. and Farimani, A. B. "Graph neural network-accelerated Lagrangian fluid simulation". In: *Computers & Graphics* 103 (2022), pp. 201–211. DOI: 10.1016/j.cag.2022.02.004.

[166] Ling, J., Kurzawski, A., and Templeton, J. "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance". In: *Journal of Fluid Mechanics* 807 (Nov. 2016), pp. 155–166. DOI: 10.1017/jfm.2016.615.

[167] Luke, S. "Two fast tree-creation algorithms for genetic programming". In: *IEEE Transactions on Evolutionary Computation* 4.3 (Sept. 2000), pp. 274–283. DOI: 10.1109/4235.873237.

[168] Luo, C., Chen, C., and Jiang, Z. "A divide and conquer method for symbolic regression". In: *International Journal of Computational Methods* 19.08 (Oct. 2022), p. 2142002. DOI: 10.1142/S0219876221420020.

[169] Makke, N. and Chawla, S. "Interpretable scientific discovery with symbolic regression: a review". In: *Artificial Intelligence Review* 57.1 (Jan. 2024), p. 2. DOI: 10.1007/s10462-023-10622-0.

[170] Martinek, V., Frotscher, O., Richter, M., and Herzog, R. *Introducing Thermodynamics-Informed Symbolic Regression – A Tool for Thermodynamic Equations of State Development*. Sept. 2023. URL: http://arxiv.org/abs/2309.02805.

[171] Martinek, V., Reuter, J., Frotscher, O., Mostaghim, S., Richter, M., and Herzog, R. *Shape Constraints in Symbolic Regression using Penalized Least Squares*. 2024. URL: https://arxiv.org/abs/2405.20800.

[172] McConaghy, T. "FFX: Fast, Scalable, Deterministic Symbolic Regression Technology". In: *Genetic Programming Theory and Practice IX*. Ed. by Riolo, R., Vladislavleva, E., and Moore, J. H. New York, NY: Springer New York, 2011, pp. 235–260. DOI: 10.1007/978-1-4614-1770-5_13.

[173] McConaghy, T. "Latent Variable Symbolic Regression for High-Dimensional Inputs". In: *Genetic Programming Theory and Practice VII*. Ed. by Riolo, R., O'Reilly, U.-M., and McConaghy, T. Boston, MA: Springer US, 2010, pp. 103–118. DOI: 10.1007/978-1-4419-1626-6_7.

[174] McConaghy, T., Palmers, P., Steyaert, M., and Gielen, G. G. E. "Trustworthy Genetic Programming-Based Synthesis of Analog Circuit Topologies Using Hierarchical Domain-Specific Building Blocks". In: *IEEE Transactions on Evolutionary Computation* 15.4 (Aug. 2011), pp. 557–570. DOI: 10.1109/TEVC.2010.2093581.

[175] McKay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., and O'Neill, M. "Grammar-based Genetic Programming: a survey". In: *Genetic Programming and Evolvable Machines* 11.3 (Sept. 2010), pp. 365–396. DOI: 10.1007/s10710-010-9109-y.

[176] Mei, Y., Nguyen, S., and Zhang, M. "Constrained Dimensionally Aware Genetic Programming for Evolving Interpretable Dispatching Rules in Dynamic Job Shop Scheduling". In: *Simulated Evolution and Learning*. Ed. by Shi, Y., Tan, K. C., Zhang, M., Tang, K., Li, X., Zhang, Q., Tan, Y., Middendorf, M., and Jin, Y. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 435–447. DOI: 10.1007/978-3-319-68759-9_36.

[177] Mengel, T., Steffanic, P., Hughes, C., Da Silva, A. C. O., and Nattrass, C. "Interpretable machine learning methods applied to jet background subtraction in heavy-ion collisions". In: *Physical Review C* 108.2 (Aug. 2023), p. L021901. DOI: 10.1103/PhysRevC.108.L021901.

[178] Miller, J. F., ed. *Cartesian Genetic Programming*. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-17310-3.

[179] Miller, J. F. and Thomson, P. "Cartesian Genetic Programming". In: *Genetic Programming*. Ed. by Goos, G., Hartmanis, J., Van Leeuwen, J., Poli, R., Banzhaf, W., Langdon, W. B., Miller, J., Nordin, P., and Fogarty, T. C. Vol. 1802. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 121–132.

[180] Mitchell, T. M. *Machine learning*. Nachdr. McGraw-Hill series in Computer Science. New York: McGraw-Hill, 1997.

[181] Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of Machine Learning*. 2nd ed. MIT Press, 2018.

[182] Molnar, C. *Interpretable machine learning: a guide for making black box models explainable*. Second edition. Munich, Germany: Christoph Molnar, 2022.

[183] Montana, D. J. "Strongly Typed Genetic Programming". In: *Evolutionary Computation* 3.2 (June 1995), pp. 199–230. DOI: 10.1162/evco.1995.3.2.199.

[184] Montesinos López, O. A., Montesinos López, A., and Crossa, J. *Multivariate Statistical Machine Learning Methods for Genomic Prediction*. Cham: Springer International Publishing, 2022. DOI: 10.1007/978-3-030-89010-0.

[185] Moore, W. C. and Balachandar, S. "Lagrangian investigation of pseudo-turbulence in multiphase flow using superposable wakes". In: *Physical Review Fluids* 4.11 (Nov. 2019), p. 114301. DOI: 10.1103/PhysRevFluids.4.114301.

[186] Moore, W. C., Balachandar, S., and Akiki, G. "A hybrid point-particle force model that combines physical and data-driven approaches". In: *Journal of Computational Physics* 385 (May 2019), pp. 187–208. DOI: 10.1016/j.jcp.2019.01.053.

[187] Moraglio, A., Krawiec, K., and Johnson, C. G. "Geometric Semantic Genetic Programming". In: *Parallel Problem Solving from Nature - PPSN XII*. Ed. by Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Coello, C. A. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M. Vol. 7491. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 21–31. DOI: 10.1007/978-3-642-32937-1_3.

[188] Mousavi Astarabadi, S. S. and Ebadzadeh, M. M. "Avoiding Overfitting in Symbolic Regression Using the First Order Derivative of GP Trees". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. Madrid Spain: ACM, July 2015, pp. 1441–1442. DOI: 10.1145/2739482.2764662.

[189] Mundhenk, T. N., Landajuela, M., Glatt, R., Santiago, C. P., Faissol, D. M., and Petersen, B. K. "Symbolic Regression via Neural-Guided Genetic Programming Population Seeding". In: *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*. Vol. 34. Advances in Neural Information Processing Systems. Curran Associates, Inc., 2021, pp. 24912–24923.

[190] Murphy, K. P. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[191] Murphy, K. P., Soliman, M., Duran-Martin, G., Kara, A., Liang, A., Reddy, S., and Patel, D. *PyProbML library for Probabilistic Machine Learning*. Aug. 2021. URL: https://github.com/probml/pyprobml.

[192] Nadizar, G., Garrow, F., Sakallioglu, B., Canonne, L., Silva, S., and Vanneschi, L. "An Investigation of Geometric Semantic GP with Linear Scaling". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Lisbon Portugal: ACM, July 2023, pp. 1165–1174. DOI: 10.1145/3583131.3590418.

[193] Nocedal, J. and Wright, S. J. *Numerical optimization*. 2nd ed. Springer series in operations research and financial engineering. New York: Springer, 2006.

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

[195] O'Neill, M., Vanneschi, L., Gustafson, S., and Banzhaf, W. "Open issues in genetic programming". In: *Genetic Programming and Evolvable Machines* 11.3-4 (Sept. 2010), pp. 339–363. DOI: 10.1007/s10710-010-9113-2.

[196] Oh, H., Amici, R., Bomarito, G., Zhe, S., Kirby, R., and Hochhalter, J. *Genetic Programming Based Symbolic Regression for Analytical Solutions to Differential Equations*. Feb. 2023. URL: http://arxiv.org/abs/2302.03175.

[197] Oltean, M. "A Comparison of Several Linear Genetic Programming Techniques". In: *Complex Systems* 14.4 (2003), pp. 285–314. DOI: 10.25088/ComplexSystems.14.4.285.

[198] Ono, K., Hanada, Y., Kumano, M., and Kimura, M. "Enhancing Island Model Genetic Programming by Controlling Frequent Trees". In: *Journal of Artificial Intelligence and Soft Computing Research* 9.1 (Jan. 2019), pp. 51–65. DOI: 10.2478/jaiscr-2018-0024.

[199] Ono, K., Hanada, Y., Kumano, M., and Kimura, M. "Island model genetic programming based on frequent trees". In: *2013 IEEE Congress on Evolutionary Computation*. June 2013, pp. 2988–2995. DOI: 10.1109/CEC.2013.6557933.

[200] Orzechowski, P., La Cava, W., and Moore, J. H. "Where are we now? A large benchmark study of recent symbolic regression methods". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. July 2018, pp. 1183–1190. DOI: 10.1145/3205455.3205539.

[201] Owen, C. A., Dick, G., and Whigham, P. A. "Characterizing Genetic Programming Error Through Extended Bias and Variance Decomposition". In: *IEEE Transactions on Evolutionary Computation* 24.6 (Dec. 2020), pp. 1164–1176. DOI: 10.1109/TEVC.2020.2990626.

[202] Owen, C. A., Dick, G., and Whigham, P. A. "Feature Standardisation in Symbolic Regression". In: *AI 2018: Advances in Artificial Intelligence*. Ed. by Mitrovic, T., Xue, B., and Li, X. Vol. 11320. Cham: Springer International Publishing, 2018, pp. 565–576. DOI: 10.1145/3377930.3390237.

[203] Pandey, P., Reuter, J., Steup, C., and Mostaghim, S. *The Road to Learning Explainable Inverse Kinematic Models: Graph Neural Networks as Inductive Bias for Symbolic Regression*. 2025. URL: https://arxiv.org/abs/2501.13641.

[204] Parker, J., Khoogar, A., and Goldberg, D. "Inverse kinematics of redundant robots using genetic algorithms". In: *Proceedings, 1989 International Conference on Robotics and Automation*. Scottsdale, AZ, USA: IEEE Comput. Soc. Press, 1989, pp. 271–276. DOI: 10.1109/ROBOT.1989.100000.

[205] Pawlak, T. P. and Krawiec, K. "Competent Geometric Semantic Genetic Programming for Symbolic Regression and Boolean Function Synthesis". In: *Evolutionary Computation* 26.2 (June 2018), pp. 177–212. DOI: 10.1162/evco_a_00205.

[206] Pawlak, T. P., Wieloch, B., and Krawiec, K. "Semantic Backpropagation for Designing Search Operators in Genetic Programming". In: *IEEE Transactions on Evolutionary Computation* 19.3 (June 2015), pp. 326–340. DOI: 10.1109/TEVC.2014.2321259.

[207] Petersen, B. K., Landajuela, M., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. *Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients*. Apr. 2021. URL: http://arxiv.org/abs/1912.04871.

[208] Pires, A. M. and Branco, J. A. *High dimensionality: The latest challenge to data analysis*. Feb. 2019. URL: http://arxiv.org/abs/1902.04679.

[209] Piringer, D., Wagner, S., Haider, C., Fohler, A., Silber, S., and Affenzeller, M. "Improving the Flexibility of Shape-Constrained Symbolic Regression with Extended Constraints". In: *Computer Aided Systems Theory – EUROCAST 2022*. Ed. by Moreno-Díaz, R., Pichler, F., and Quesada-Arencibia, A. Vol. 13789. Cham: Springer Nature Switzerland, 2022, pp. 155–163. DOI: 10.1007/978-3-031-25312-6_18.

[210] Poli, R., Langdon, W. B., McPhee, N. F., and Koza, J. R. *A field guide to genetic programming*. Morrisville, NC: Lulu Press, 2008.

[211] Powers, S., Smith, J., and Pinciroli, C. "Extracting Symbolic Models of Collective Behaviors with Graph Neural Networks and Macro-Micro Evolution". In: *Swarm Intelligence*. Ed. by Dorigo, M., Hamann, H., López-Ibáñez, M., García-Nieto, J., Engelbrecht, A., Pinciroli, C., Strobel, V., and Camacho-Villalón, C. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 142–154. DOI: 10.1007/978-3-031-20176-9_12.

[212] Punch, W. F. "How effective are multiple populations in genetic programming". In: *Genetic programming 1998: Proceedings of the third annual conference*. Ed. by Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H., and Riolo, R. University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann, 1998, pp. 308–313.

[213] Radwan, Y. A., Kronberger, G., and Winkler, S. *A Comparison of Recent Algorithms for Symbolic Regression to Genetic Programming*. June 2024. URL: http://arxiv.org/abs/2406.03585.

[214] Raissi, M., Perdikaris, P., and Karniadakis, G. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (Feb. 2019), pp. 686–707. DOI: 10.1016/j.jcp.2018.10.045.

[215] Raissi, M., Yazdani, A., and Karniadakis, G. E. "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations". In: *Science* 367.6481 (Feb. 2020), pp. 1026–1030. DOI: 10.1126/science.aaw4741.

[216] Randall, D. L., Townsend, T. S., Hochhalter, J. D., and Bomarito, G. F. "Bingo: a customizable framework for symbolic regression with genetic programming". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. Boston Massachusetts: ACM, July 2022, pp. 2282–2288. DOI: 10.1145/3520304.3534031.

[217] Rasmussen, C. E. and Williams, C. K. *Gaussian Processes for Machine Learning*. Cambridge: The MIT Press, 2019.

[218] Ratle, A. and Sebag, M. "Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery". In: *Parallel Problem Solving from Nature PPSN VI*. Ed. by Goos, G., Hartmanis, J., Van Leeuwen, J., Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P. Vol. 1917. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 211–220. DOI: 10.1007/3-540-45356-3_21.

[219] Ratle, A. and Sebag, M. "Grammar-guided genetic programming and dimensional consistency: application to non-parametric identification in mechanics". In: *Applied Soft Computing* 1.1 (June 2001), pp. 105–118. DOI: 10.1016/S1568-4946(01)00009-6.

[220] Reinbold, P. A. K., Kageorge, L. M., Schatz, M. F., and Grigoriev, R. O. "Robust learning from noisy, incomplete, high-dimensional experimental data via physically constrained symbolic regression". In: *Nature Communications* 12.1 (May 2021), p. 3219. DOI: 10.1038/s41467-021-23479-0.

[221] Reuter, J. "Genetic Programming - Based Inverse Kinematics for Robotic Manipulators". MA thesis. Otto-von-Guericke-University Magdeburg, 2021.

[222] Reuter, J., Cendrollu, M., Evrard, F., Mostaghim, S., and Wachem, B. van. "Towards Improving Simulations of Flows around Spherical Particles Using Genetic Programming". In: *2022 IEEE Congress on Evolutionary Computation (CEC)*. July 2022, pp. 1–8. DOI: 10.1109/CEC55065.2022.9870301.

[223] Reuter, J., Elmestikawy, H., Evrard, F., Mostaghim, S., and Wachem, B. van. "Graph Networks as Inductive Bias for Genetic Programming: Symbolic Models for Particle-Laden Flows". In: *Genetic Programming*. Ed. by Pappa, G., Giacobini, M., and Vasicek, Z. Cham: Springer Nature Switzerland, 2023, pp. 36–51. DOI: 10.1007/978-3-031-29573-7_3.

[224] Reuter, J., Martinek, V., Herzog, R., and Mostaghim, S. "Unit-Aware Genetic Programming for the Development of Empirical Equations". In: *Parallel Problem Solving from Nature – PPSN XVIII*. Ed. by Affenzeller, M., Winkler, S. M., Kononova, A. V., Trautmann, H., Tušar, T., Machado, P., and Bäck, T. Vol. 15148. Cham: Springer Nature Switzerland, 2024, pp. 168–183. DOI: 10.1007/978-3-031-70055-2_11.

[225] Reuter, J., Pandey, P., and Mostaghim, S. "Multi-Objective Island Model Genetic Programming for Predicting the Stokes Flow around a Sphere". In: *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*. Mexico City, Mexico: IEEE, Dec. 2023, pp. 1485–1490. DOI: 10.1109/SSCI52147.2023.10371955.

[226] Reuter, J., Steup, C., and Mostaghim, S. "Genetic Programming - Based Inverse Kinematics for Robotic Manipulators". In: *Genetic Programming*. Ed. by Medvet, E., Pappa, G., and Xue, B. Vol. 13223. Cham: Springer International Publishing, 2022, pp. 130–145. DOI: 10.1007/978-3-031-02056-8_9.

[227] Rhinehart, R. R. *Nonlinear regression modeling for engineering applications: modeling, model validation, and enabling design of experiments*. Wiley-ASME Press Series. Chichester: John Wiley; ASME Press, 2016.

[228] Richardson, J. F. and Zaki, W. N. "The sedimentation of a suspension of uniform spheres under conditions of viscous flow". In: *Chemical Engineering Science* 3.2 (Apr. 1954), pp. 65–73. DOI: 10.1016/0009-2509(54)85015-9.

[229] Ritz, C. and Streibig, J. C., eds. *Nonlinear Regression with R*. New York, NY: Springer New York, 2009. DOI: 10.1007/978-0-387-09616-2.

[230] Rockett, P. "Constant optimization and feature standardization in multiobjective genetic programming". In: *Genetic Programming and Evolvable Machines* 23.1 (Mar. 2022), pp. 37–69. DOI: 10.1007/s10710-021-09410-y.

[231] Rodrigues, N. M., Batista, J. E., La Cava, W., Vanneschi, L., and Silva, S. "SLUG: Feature Selection Using Genetic Algorithms and Genetic Programming". In: *Genetic Programming*. Ed. by Medvet, E., Pappa, G., and Xue, B. Vol. 13223. Cham: Springer International Publishing, 2022, pp. 68–84. DOI: 10.1007/978-3-031-02056-8_5.

[232] Rodriguez-Coayahuitl, L., Morales-Reyes, A., Escalante, H. J., and Coello Coello, C. A. "Cooperative Co-Evolutionary Genetic Programming for High Dimensional Problems". In: *Parallel Problem Solving from Nature – PPSN XVI*. Ed. by Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M., and Trautmann, H. Vol. 12270. Cham: Springer International Publishing, 2020, pp. 48–62. DOI: doi.org/10.1007/978-3-030-58115-2_4.

[233] Rosenblatt, F. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[234] Ross, A., Li, Z., Perezhogin, P., Fernandez-Granda, C., and Zanna, L. "Benchmarking of Machine Learning Ocean Subgrid Parameterizations in an Idealized Model". In: *Journal of Advances in Modeling Earth Systems* 15.1 (Jan. 2023), e2022MS003258. DOI: 10.1029/2022MS003258.

[235] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0.

[236] Russeil, E., França, F. O. de, Malanchev, K., Burlacu, B., Ishida, E., Leroux, M., Michelin, C., Moinard, G., and Gangler, E. "Multiview Symbolic Regression". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Melbourne VIC Australia: ACM, July 2024, pp. 961–970. DOI: 10.1145/3638529.3654087.

[237] Russell, S. J. and Norvig, P. *Artificial intelligence: a modern approach*. Third edition, Global edition. Prentice Hall series in artificial intelligence. Pearson Education, 2016.

[238] Ryan, C., Collins, J., and Neill, M. O. "Grammatical evolution: Evolving programs for an arbitrary language". In: *Genetic Programming*. Ed. by Goos, G., Hartmanis, J., Van Leeuwen, J., Banzhaf, W., Poli, R., Schoenauer, M., and Fogarty, T. C. Vol. 1391. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 83–96.

[239] Ryan, C. and Keijzer, M. "An Analysis of Diversity of Constants of Genetic Programming". In: *Genetic Programming*. Ed. by Goos, G., Hartmanis, J., Van Leeuwen, J., Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., and Costa, E. Vol. 2610. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 404–413. DOI: 10.1007/3-540-36599-0_38.

[240] Al-Sahaf, H., Song, A., Neshatian, K., and Zhang, M. "Two-Tier genetic programming: towards raw pixel-based image classification". In: *Expert Systems with Applications* 39.16 (Nov. 2012), pp. 12291–12301. DOI: 10.1016/j.eswa.2012.02.123.

[241] Sahli Costabal, F., Pezzuto, S., and Perdikaris, P. " -PINNs: Physics-informed neural networks on complex geometries". In: *Engineering Applications of Artificial Intelligence* 127 (Jan. 2024), p. 107324. DOI: 10.1016/j.engappai.2023.107324.

[242] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. W. *Learning to Simulate Complex Physics with Graph Networks*. Sept. 2020. URL: http://arxiv.org/abs/2002.09405.

[243] Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. "Graph networks as learnable physics engines for inference and control". In: *International conference on machine learning*. PMLR, 2018, pp. 4470–4479.

[244] Scarselli, F., Gori, M., Ah Chung Tsoi, Hagenbuchner, M., and Monfardini, G. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (Jan. 2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

[245] Schmidt, M. and Lipson, H. "Age-Fitness Pareto Optimization". In: *Genetic Programming Theory and Practice VIII*. Ed. by Riolo, R., McConaghy, T., and Vladislavleva, E. Vol. 8. New York, NY: Springer New York, 2011, pp. 129–146. DOI: 10.1007/978-1-4419-7747-2_8.

[246] Schmidt, M. and Lipson, H. "Distilling Free-Form Natural Laws from Experimental Data". In: *Science* 324.5923 (Apr. 2009), pp. 81–85. DOI: 10.1126/science.1165893.

[247] Schmidt, M. D. and Lipson, H. "Incorporating expert knowledge in evolutionary search: a study of seeding methods". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. Montreal Québec Canada: ACM, July 2009, pp. 1091–1098. DOI: 10.1145/1569901.1570048.

[248] Schmidt, M. D. and Lipson, H. "Learning noise". In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. London England: ACM, July 2007, pp. 1680–1685. DOI: 10.1145/1276958.1277289.

[249] Schmitt, J., Kuckuk, S., and Köstler, H. "EvoStencils: a grammar-based genetic programming approach for constructing efficient geometric multigrid methods". In: *Genetic Programming and Evolvable Machines* 22.4 (Dec. 2021), pp. 511–537. DOI: 10.1007/s10710-021-09412-w.

[250] Schoenauer, M. and Sebag, M. *Using Domain Knowledge in Evolutionary System Identification*. Feb. 2006. URL: http://arxiv.org/abs/cs/0602021.

[251] Scholkopf, B. and Smola, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, Ipswich: MIT Press ; Ebsco Publishing [distributor], 2018.

[252] Schwab, I., Senn, M., and Link, N. "Improving Expert Knowledge in Dynamic Process Monitoring by Symbolic Regression". In: *2012 Sixth International Conference on Genetic and Evolutionary Computing*. Kitakyushu, Japan: IEEE, Aug. 2012, pp. 132–135. DOI: 10.1109/ICGEC.2012.105.

[253] Searson, D. P., Leahy, D. E., and Willis, M. J. "GPTIPS:An Open Source Genetic Programming Toolbox For Multigene Symbolic Regression". In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 1. 2010, pp. 77–80.

[254] Seyed-Ahmadi, A. and Wachs, A. "Microstructure-informed probability-driven point-particle model for hydrodynamic forces and torques in particle-laden flows". In: *Journal of Fluid Mechanics* 900 (Oct. 2020), A21. DOI: 10.1017/jfm.2020.453.

[255] Seyed-Ahmadi, A. and Wachs, A. "Physics-inspired architecture for neural network modeling of forces and torques in particle-laden flows". In: *Computers & Fluids* 238 (Apr. 2022), p. 105379. DOI: 10.1016/j.compfluid.2022.105379.

[256] Shao, H., Villaescusa-Navarro, F., Genel, S., Spergel, D. N., Anglés-Alcázar, D., Hernquist, L., Davé, R., Narayanan, D., Contardo, G., and Vogelsberger, M. "Finding Universal Relations in Subhalo Properties with Artificial Intelligence". In: *The Astrophysical Journal* 927.1 (Mar. 2022), p. 85. DOI: 10.3847/1538-4357/ac4d30.

[257] El-Sherbiny, A., Elhosseini, M. A., and Haikal, A. Y. "A comparative study of soft computing methods to solve inverse kinematics problem". In: *Ain Shams Engineering Journal* 9.4 (Dec. 2018), pp. 2535–2548. DOI: 10.1016/j.asej.2017.08.001.

[258] Smits, G. F. and Kotanchek, M. "Pareto-Front Exploitation in Symbolic Regression". In: *Genetic Programming Theory and Practice II*. Ed. by O'Reilly, U.-M., Yu, T., Riolo, R., and Worzel, B. Vol. 8. New York: Springer-Verlag, 2005, pp. 283–299. DOI: 10.1007/0-387-23254-0_17.

[259] Sobania, D., Briesch, M., Wittenberg, D., and Rothlauf, F. "Analyzing optimized constants in genetic programming on a real-world regression problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. Boston Massachusetts: ACM, July 2022, pp. 606–607. DOI: 10.1145/3520304.3528896.

[260] Spector, L. "Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report". In: *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. Philadelphia Pennsylvania USA: ACM, July 2012, pp. 401–408. DOI: 10.1145/2330784.2330846.

[261] Srinivas, N. and Deb, K. "Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms". In: *Evolutionary Computation* 2.3 (Sept. 1994), pp. 221–248. DOI: 10.1162/evco.1994.2.3.221.

[262] Srisuk, P., Sento, A., and Kitjaidure, Y. "Forward kinematic-like neural network for solving the 3D reaching inverse kinematics problems". In: *2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. Phuket: IEEE, June 2017, pp. 214–217. DOI: 10.1109/ECTICon.2017.8096211.

[263] Stokes, G. G. "On the Effect of the Internal Friction of Fluids on the Motion of Pendulums". In: *Transactions of the Cambridge Philosophical Society* 9 (Jan. 1851), p. 8.

[264] Sun, H. and Moscato, P. "A Memetic Algorithm for Symbolic Regression". In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. Wellington, New Zealand: IEEE, June 2019, pp. 2167–2174. DOI: 10.1109/CEC.2019.8789889.

[265] Sutton, R. S. and Barto, A. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning. Cambridge, Massachusetts London, England: The MIT Press, 2020.

[266] Tanese, R. "Distributed Genetic Algorithms for Function Optimization". PhD thesis. Ann Arb or, MI: University of Michigan, 1989.

[267] Tenachi, W., Ibata, R., and Diakogiannis, F. I. "Deep Symbolic Regression for Physics Guided by Units Constraints: Toward the Automated Discovery of Physical Laws". In: *The Astrophysical Journal* 959.2 (Dec. 2023), p. 99. DOI: 10.3847/1538-4357/ad014c.

[268] Tenneti, S., Garg, R., and Subramaniam, S. "Drag law for monodisperse gas–solid systems using particle-resolved direct numerical simulation of flow past fixed assemblies of spheres". In: *International Journal of Multiphase Flow* 37.9 (Nov. 2011), pp. 1072–1092. DOI: 10.1016/j.ijmultiphaseflow.2011.05.010.

[269] Topchy, A. and Punch, W. F. "Faster Genetic Programming based on Local Gradient Search of Numeric Leaf Values". In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. GECCO'01. Morgan Kaufmann Publishers Inc., 2001, pp. 155–162.

[270] Tran, B., Xue, B., and Zhang, M. "Genetic programming for feature construction and selection in classification on high-dimensional data". In: *Memetic Computing* 8.1 (Mar. 2016), pp. 3–15. DOI: 10.1007/s12293-015-0173-y.

[271] Udrescu, S.-M., Tan, A., Feng, J., Neto, O., Wu, T., and Tegmark, M. "AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity". In: *Advances in Neural Information Processing Systems*. Ed. by Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. Vol. 33. Curran Associates, Inc., 2020, pp. 4860–4871.

[272] Udrescu, S.-M. and Tegmark, M. "AI Feynman: A physics-inspired method for symbolic regression". In: *Science Advances* 6.16 (Apr. 2020), eaay2631. DOI: 10.1126/sciadv.aay2631.

[273] Uy, N. Q., O'Neill, M., Hoai, N. X., Mckay, B., and Galván-López, E. "Semantic Similarity Based Crossover in GP: The Case for Real-Valued Function Regression". In: *Artifical Evolution*. Ed. by Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Collet, P., Monmarché, N., Legrand, P., Schoenauer, M., and Lutton, E. Vol. 5975. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 170–181. DOI: 10.1007/978-3-642-14156-0_15.

[274] Vanneschi, L. "SLIM_GSGP: The Non-bloating Geometric Semantic Genetic Programming". In: *Genetic Programming*. Ed. by Giacobini, M., Xue, B., and Manzoni, L. Vol. 14631. Cham: Springer Nature Switzerland, 2024, pp. 125–141. DOI: 10.1007/978-3-031-56957-9_8.

[275] Vanneschi, L., Castelli, M., and Silva, S. "A survey of semantic methods in genetic programming". In: *Genetic Programming and Evolvable Machines* 15.2 (June 2014), pp. 195–214. DOI: 10.1007/s10710-013-9210-0.

[276] Vanneschi, L., Castelli, M., and Silva, S. "Measuring bloat, overfitting and functional complexity in genetic programming". In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. Portland Oregon USA: ACM, July 2010, pp. 877–884. DOI: 10.1145/1830483.1830643.

[277] Vapnik, V., Golowich, S., and Smola, A. "Support vector method for function approximation, regression estimation and signal processing". In: *Advances in neural information processing systems*. Ed. by Mozer, M., Jordan, M., and Petsche, T. Vol. 9. MIT Press, 1996.

[278] Versino, D., Tonda, A., and Bronkhorst, C. A. "Data driven modeling of plastic deformation". In: *Computer Methods in Applied Mechanics and Engineering* 318 (May 2017), pp. 981–1004. DOI: 10.1016/j.cma.2017.02.016.

[279] Verstyuk, S. and Douglas, M. R. "Machine Learning the Gravity Equation for International Trade". In: *SSRN Electronic Journal* (2022). DOI: 10.2139/ssrn.4053795.

[280] Virgolin, M., Alderliesten, T., Witteveen, C., and Bosman, P. A. N. "Improving Model-Based Genetic Programming for Symbolic Regression of Small Expressions". In: *Evolutionary Computation* 29.2 (June 2021), pp. 211–237. DOI: 10.1162/evco_a_00278.

[281] Virgolin, M., De Lorenzo, A., Medvet, E., and Randone, F. "Learning a Formula of Interpretability to Learn Interpretable Formulas". In: *Parallel Problem Solving from Nature – PPSN XVI*. Ed. by Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M., and Trautmann, H. Vol. 12270. Cham: Springer International Publishing, 2020, pp. 79–93. DOI: 10.1007/978-3-030-58115-2_6.

[282] Virgolin, M. and Pissis, S. P. *Symbolic Regression is NP-hard*. July 2022. URL: http://arxiv.org/abs/2207.01018.

[283] Vladislavleva, E., Smits, G., and Den Hertog, D. "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 (Apr. 2009), pp. 333–349. DOI: 10.1109/TEVC.2008.926486.

[284] Von Preetzmann, N., Kleinrahm, R., Eckmann, P., Cavuoto, G., and Richter, M. "Density Measurements of an Air-Like Binary Mixture over the Temperature Range from 100 K to 298.15 K at Pressures up to 8.0 MPa". In: *International Journal of Thermophysics* 42.9 (Sept. 2021), p. 127. DOI: 10.1007/s10765-021-02871-4.

[285] Wagaa, N., Kallel, H., and Mellouli, N. "Analytical and deep learning approaches for solving the inverse kinematic problem of a high degrees of freedom robotic arm". In: *Engineering Applications of Artificial Intelligence* 123 (Aug. 2023), p. 106301. DOI: 10.1016/j.engappai.2023.106301.

[286] Wang, S. and Gupta, M. "Deontological ethics by monotonicity shape constraints". In: *Proceedings of the twenty third international conference on artificial intelligence and statistics*. Ed. by Chiappa, S. and Calandra, R. Vol. 108. Proceedings of machine learning research. PMLR, Aug. 2020, pp. 2043–2054.

[287] Wappler, S. and Wegener, J. "Evolutionary Unit Testing Of Object-Oriented Software Using A Hybrid Evolutionary Algorithm". In: *2006 IEEE International Conference on Evolutionary Computation*. July 2006, pp. 851–858. DOI: 10.1109/CEC.2006.1688400.

[288] Watson, J., Song, C., Weeger, O., Gruner, T., Le, A. T., Hansel, K., Hendawy, A., Arenz, O., Trojak, W., Cranmer, M., D'Eramo, C., Bülow, F., Goyal, T., Peters, J., and Hoffman, M. W. *Machine Learning with Physics Knowledge for Prediction: A Survey*. Aug. 2024. URL: http://arxiv.org/abs/2408.09840.

[289] Wei, Y., Jian, S., He, S., and Wang, Z. "General approach for inverse kinematics of nR robots". In: *Mechanism and Machine Theory* 75 (May 2014), pp. 97–106. DOI: 10.1016/j.mechmachtheory.2014.01.008.

[290] Whigham, P. A. "Grammatically-based genetic programming". In: *Proceedings of the workshop on genetic programming: From theory to real-world applications*. Ed. by Rosca, J. P. Tahoe City, California, USA, July 1995, pp. 33–41.

[291] Whigham, P. "Inductive bias and genetic programming". In: *1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*. Vol. 1995. Sheffield, UK: IEE, 1995, pp. 461–466. DOI: 10.1049/cp:19951092.

[292] Willis, M.-J. "Genetic programming: an introduction and survey of applications". In: *Second International Conference on Genetic Algorithms in Engineering Systems*. Vol. 1997. Glasgow, UK: IEE, 1997, pp. 314–319. DOI: 10.1049/cp:19971199.

[293] Wilson, R. J. *Introduction to graph theory*. 4. ed. Harlow Munich: Prentice Hall, 2009.

[294] Wolpert, D. H. "The Lack of A Priori Distinctions Between Learning Algorithms". In: *Neural Computation* 8.7 (Oct. 1996), pp. 1341–1390. DOI: 10.1162/neco.1996.8.7.1341.

[295] Xie, H. "An Analysis of Selection in Genetic Programming". PhD thesis. Te Herenga Waka - Victoria University of Wellington, 2009.

[296] Zeng, P., Song, X., Lensen, A., Ou, Y., Sun, Y., Zhang, M., and Lv, J. *Differentiable Genetic Programming for High-dimensional Symbolic Regression*. Apr. 2023. URL: http://arxiv.org/abs/2304.08915.

[297] Zhang, Y. and Rockett, P. "A Comparison of three evolutionary strategies for multiobjective genetic programming". In: *Artificial Intelligence Review* 27.2-3 (Mar. 2007), pp. 149–163. DOI: 10.1007/s10462-008-9093-2.

[298] Zhong, L., Zhong, J., and Lu, C. "A Comparative Analysis of Dimensionality Reduction Methods for Genetic Programming to Solve High-Dimensional Symbolic Regression Problems". In: *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Melbourne, Australia: IEEE, Oct. 2021, pp. 476–483. DOI: 10.1109/SMC52423.2021.9658595.

[299] Zille, H., Evrard, F., Reuter, J., Mostaghim, S., and Wachem, B. "Assessment of Multi-objective and Coevolutionary Genetic Programming for Predicting the Stokes Flow around a Sphere". In: *14th International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control*. Jan. 2021, p. 190. DOI: 10.7712/140121.7959.18341.

[300] Zitzler, E. "Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications". PhD thesis. Zürich, Switzerland: Swiss Federal Institute of Technology (ETH), 1999.

Acronyms

ADF	automatically defined functions
ALPS	age-layered population structure
ANN	artificial neural network
BFGS	Broyden–Fletcher–Goldfarb–Shanno
CCGP	cooperative coevolutionary genetic programming
CO-NLS	constant optimization with nonlinear least squares
D&C	divide and conquer
D-H	Denavit-Hartenberg
DOF	degrees of freedom
DSR	deep symbolic regression
EA	evolutionary algorithm
EC	evolutionary computation
ERC	ephemeral random constant
FEAT	feature engineering automation tool
FFX	fast function extraction
FK	forward kinematics
GNN	graph neural network
GP	genetic programming
GSGP	geometric semantic genetic programming
HFC	hierarchical fair competition
HOF	Hall of Fame
IA	interval arithmetic
IK	inverse kinematics
IK-CCGP	cooperative coevolutionary genetic programming for inverse kinematics
IM	island model
IMGP	island model genetic programming
ITEA	interaction transformation evolutionary algorithm
MAE	mean absolute error
MaxAE	maximum absolute error
ML	machine learning

MLP	multilayer perceptron
MOEA	multi-objective evolutionary algorithms
MOO	multi-objective optimization
MPNN	message passing neural network
MSE	mean squared error
NSGA	non-dominated sorting algorithm
PDE	partial differential equation
PIEP	pairwise interaction extended point-particle
PIML	physics-informed machine learning
PINN	physics-informed neural network
PO	Pareto-optimal
PR-DNS	particle-resolved direct numerical simulation
$\mathbf{R}^2$	coefficient of determination
RL	reinforceement learning
RMSE	rooted mean squared error
RNN	recurrent neural network
SINDy	sparse identification of nonlinear dynamics
SIP	superimposition method
SR	symbolic regression
SVR	support vector regression
VFEL	volume filtered Euler Lagrange

Author's Publications

[79] Elmestikawy, H., Reuter, J., Evrard, F., Mostaghim, S., and Van Wachem, B. "Deterministic drag modelling for spherical particles in Stokes regime using data-driven approaches". In: *International Journal of Multiphase Flow* 178 (Aug. 2024), p. 104880. DOI: 10.1016/j.ijmultiphaseflow.2024.104880.

[171] Martinek, V., Reuter, J., Frotscher, O., Mostaghim, S., Richter, M., and Herzog, R. *Shape Constraints in Symbolic Regression using Penalized Least Squares*. 2024. DOI: 10.48550/ARXIV.2405.20800.

[203] Pandey, P., Reuter, J., Steup, C., and Mostaghim, S. *The Road to Learning Explainable Inverse Kinematic Models: Graph Neural Networks as Inductive Bias for Symbolic Regression*. 2025. DOI: 10.48550/arXiv.2501.13641.

[221] Reuter, J. "Genetic Programming - Based Inverse Kinematics for Robotic Manipulators". MA thesis. Otto-von-Guericke-University Magdeburg, 2021.

[222] Reuter, J., Cendrollu, M., Evrard, F., Mostaghim, S., and Wachem, B. van. "Towards Improving Simulations of Flows around Spherical Particles Using Genetic Programming". In: *2022 IEEE Congress on Evolutionary Computation (CEC)*. July 2022, pp. 1–8. DOI: 10.1109/CEC55065.2022.9870301.

[223] Reuter, J., Elmestikawy, H., Evrard, F., Mostaghim, S., and Wachem, B. van. "Graph Networks as Inductive Bias for Genetic Programming: Symbolic Models for Particle-Laden Flows". In: *Genetic Programming*. Ed. by Pappa, G., Giacobini, M., and Vasicek, Z. Cham: Springer Nature Switzerland, 2023, pp. 36–51. DOI: 10.1007/978-3-031-29573-7_3.

[224] Reuter, J., Martinek, V., Herzog, R., and Mostaghim, S. "Unit-Aware Genetic Programming for the Development of Empirical Equations". In: *Parallel Problem Solving from Nature – PPSN XVIII*. ed. by Affenzeller, M., Winkler, S. M., Kononova, A. V., Trautmann, H., Tušar, T., Machado, P., and Bäck, T. Vol. 15148. Cham: Springer Nature Switzerland, 2024, pp. 168–183. DOI: 10.1007/978-3-031-70055-2_11.

[225] Reuter, J., Pandey, P., and Mostaghim, S. "Multi-Objective Island Model Genetic Programming for Predicting the Stokes Flow around a Sphere". In: *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*. Mexico City, Mexico: IEEE, Dec. 2023, pp. 1485–1490. DOI: 10.1109/SSCI52147.2023.10371955.

[226] Reuter, J., Steup, C., and Mostaghim, S. “Genetic Programming - Based Inverse Kinematics for Robotic Manipulators”. In: *Genetic Programming*. Ed. by Medvet, E., Pappa, G., and Xue, B. Vol. 13223. Cham: Springer International Publishing, 2022, pp. 130–145. DOI: 10.1007/978-3-031-02056-8_9.

[299] Zille, H., Evrard, F., Reuter, J., Mostaghim, S., and Wachem, B. “Assessment of Multi-objective and Coevolutionary Genetic Programming for Predicting the Stokes Flow around a Sphere”. In: *14th International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control*. Jan. 2021, p. 190. DOI: 10.7712/140121.7959.18341.

List of Figures

List of Tables

A Additional Equations

This appendix provides additional equations from the final experiments on the IK problem presented in Chapter 8.

msg	Compl.	Equation	R^2
1	25	$-0.744\theta_{r,\text{off}} + 0.986\theta_{r,\text{ref}} + \frac{2\theta_{s,\text{off}}}{2\Psi} + \cos(\alpha_s) + \arctan\left(-1.15, \frac{2\Phi}{2\Psi}\right)$	0.845
1	20	$-2.28 + \theta_{r,\text{ref}} + 1.0152\theta_{s,\text{off}} + \frac{\theta_{s,\text{off}} + \frac{0.162\Phi}{l_r}}{\Psi}$	0.837
1	15	$-1.09 + \theta_{r,\text{ref}} - \alpha_s + 0.2988\Psi\left(\theta_{s,\text{off}} + \Phi\right)$	0.793
2	25	$-2.21 + 0.90324\left(\alpha_r + \theta_{r,\text{ref}}\right) + \frac{0.20381\theta_{s,\text{off}}\theta_{r,\text{ref}} + 0.0704\theta_{s,\text{off}}^2\theta_{r,\text{ref}} - 0.0352\alpha_s\theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi} - 0.156\alpha_r\alpha_s + 0.312\alpha_r\theta_{s,\text{off}} - 0.156\alpha_s\theta_{r,\text{ref}} + 0.312\theta_{s,\text{off}}\theta_{r,\text{ref}}$	0.830
2	23	$-2.75 + \left(\alpha_r + \theta_{r,\text{ref}}\right)\arctan\left(1.67 + \alpha_r + 2\theta_{r,\text{off}}, \alpha_s - 0.404\Psi + 2\theta_{r,\text{off}}\right)$	0.823
2	31	$-8.5945 + \alpha_r + \theta_{r,\text{ref}} + 10.167\theta_{s,\text{off}} - 4.15\arctan\left(-17.4 + 2\Phi + 4\Psi\theta_{s,\text{off}}, \theta_{r,\text{ref}}\right) + 6.64\theta_{s,\text{off}}\arctan\left(-17.4 + 2\Phi + 4\Psi\theta_{s,\text{off}}, \theta_{r,\text{ref}}\right)$	0.862
3	28	$-\theta_{s,\text{off}} + 4.77\alpha_r + \frac{2.82 + 5.71\theta_{s,\text{off}} - 6.13\alpha_r - 1.62\theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi} - \alpha_r\theta_{r,\text{ref}}$	0.767
3	26	$-1.5045\theta_{s,\text{off}} + \frac{-1.36715\theta_{s,\text{off}}\theta_{r,\text{ref}} - 1.6805\alpha_r\theta_{s,\text{off}} - 0.23961\alpha_r\theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi} + \frac{-1.53615\alpha_r\theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi^2} - 0.21325\theta_{s,\text{off}}\theta_{r,\text{ref}} + 2.94\alpha_r\theta_{s,\text{off}} + 2.6166\Psi\theta_{s,\text{off}}$	0.748
3	24	$-2.9302\theta_{s,\text{off}} - 0.16136\Psi^2\theta_{s,\text{off}} + 3.5394\Psi\theta_{s,\text{off}} + 0.03056\Psi^2\theta_{s,\text{off}}\theta_{r,\text{ref}} - 0.644\Psi\theta_{s,\text{off}}\theta_{r,\text{ref}} - 4.368l_r\Psi\theta_{s,\text{off}} + 5.0688\Psi^2 l_r\theta_{s,\text{off}} - 0.96\Psi^2 l_r\theta_{s,\text{off}}\theta_{r,\text{ref}}$	0.745
4	17	$-0.59321\theta_{r,\text{ref}} - 1.1864\theta_{s,\text{off}} + 4.33l_r + \frac{-0.29661\theta_{r,\text{ref}} - 0.5932\Phi}{\Psi}$	0.590
4	26	$37.6\arctan\left(-0.015564\theta_{r,\text{ref}} + 0.01864\theta_{r,\text{off}} - 0.031129\theta_{s,\text{off}} + \frac{-0.007782\theta_{r,\text{ref}} - 0.015564\Phi}{\Psi}, 1.09 + \alpha_r\right)$	0.591
4	26	$0.676 + \frac{-0.284\theta_{s,\text{off}}}{l_r} - 0.0942\theta_{r,\text{ref}}^2 + \frac{-0.985 - 0.613\Phi}{\Psi}$	0.631
5	22	$\alpha_r - 1.05\theta_{r,\text{ref}} + 2\theta_{r,\text{off}} + \frac{-0.24523\theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi} - 0.0786\theta_{r,\text{ref}}^2\theta_{s,\text{off}}$	0.856
5	18	$\alpha_r - \theta_{r,\text{ref}} + 2\theta_{r,\text{off}} - 0.0886\alpha_r\theta_{s,\text{off}} + 0.1772\theta_{s,\text{off}}\Phi - 0.0886\theta_{r,\text{ref}}^2\theta_{s,\text{off}}$	0.854
5	17	$\alpha_r - \theta_{r,\text{ref}} + 2\theta_{r,\text{off}} - 0.42714\theta_{s,\text{off}}\theta_{r,\text{ref}} + 0.0756\theta_{s,\text{off}}\theta_{r,\text{ref}}\Phi$	0.841
6	22	$-0.2496 + 1.12\theta_{r,\text{ref}} - 1.6409\theta_{s,\text{off}} - 2.24\theta_{r,\text{off}} + \frac{-2.4768\theta_{s,\text{off}} - 1.92\theta_{r,\text{off}}\theta_{s,\text{off}} + 0.96\theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi} + 0.636\theta_{s,\text{off}}\theta_{r,\text{ref}} - 1.272\theta_{r,\text{off}}\theta_{s,\text{off}}$	0.876
6	18	$-3.35 + \theta_{r,\text{ref}} + \frac{-0.7928\theta_{s,\text{off}} + 0.41\theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi} + 0.82\theta_{s,\text{off}}\theta_{r,\text{ref}}$	0.845
6	21	$-0.242 + 1.12\theta_{r,\text{ref}} - 1.6484\theta_{s,\text{off}} - 2.24\theta_{r,\text{off}} + \frac{-2.6\theta_{s,\text{off}} - 2\theta_{r,\text{off}}\theta_{s,\text{off}} + \theta_{s,\text{off}}\theta_{r,\text{ref}}}{\Psi} + 0.634\theta_{s,\text{off}}\theta_{r,\text{ref}} - 1.268\theta_{r,\text{off}}\theta_{s,\text{off}}$	0.876

Table A.1: Additional equations for the six message elements of the 3 DOF manipulators. Extension to Tab. 8.10.

msg	Compl.	Equation	R^2
1	28	$0.464 + \alpha_s + 2\Theta\left(-0.584 + 1.0787 l_r - 0.586\theta_{s,\text{off}}\right) + \sin\left(1.58 - \alpha_r - \alpha_s\right)$	0.676
1	18	$0.882 + \alpha_s - 0.278\left(2\Theta + 2\theta_{s,\text{off}}\left(-0.7 + \alpha_r + 2\Theta\right)\right)$	0.645
1	21	$0.874 + \alpha_s - 0.357\left(2\Theta + 1.488\theta_{s,\text{off}}\left(-0.715 + \alpha_r + 2\Theta\right)\right)$	0.649
2	23	$-0.0787\left(-5.4 + l_s\left(14 + 2\theta_{s,\text{off}}\left(5.63 + 2\Phi\right)\right)\right)\arctan\left(0.163, 2\Psi\right)$	0.502
2	27	$\frac{0.087719\left(\alpha_r + 2\theta_{s,\text{off}}\right)}{-3.3 + \arctan\left(\frac{-0.0254}{-3.2 - 0.1274\Phi + 2\theta_{r,\text{off}} + 2\theta_{s,\text{off}}}, 2\Psi\right)}$	0.522
2	24	$-0.446\theta_{s,\text{off}} - \cos\left(-\alpha_s + \arctan\left(\left(0.478\Psi - \alpha_r\right)\left(-0.175 + 2\theta_{s,\text{off}}\right), -0.187\right)\right)$	0.478
3	27	$0.151\left(-4.89 + 2\Psi - 2\theta_{s,\text{off}}\left(0.615 + \arctan\left(0.02716\Psi\theta_{s,\text{off}}, 0.00105 - \alpha_r^2\right)\right)\right)$	0.520
3	28	$-0.929 - 0.323\left(-\alpha_r + \sin\left(2\Psi\right) + \arctan\left(0.095238\theta_{s,\text{off}}, 2\Psi\left(-3.15 - 2\theta_{s,\text{off}}\right)\right)\right)$	0.472
3	25	$\arctan\left(-4.32 - e^{2\Psi\left(-0.235 + \alpha_r - 16.026\alpha_s + 16.22\theta_{s,\text{off}}\right)}, 13.8\right)$	0.432
4	21	$\alpha_r - 3.17 l_s + \arctan\left(5.99 - 4\theta_{s,\text{off}}, 3.25 - 2\left(\Psi + \theta_{s,\text{off}}\right)\right)$	0.674
4	22	$\alpha_r + 0.792\arctan\left(6.17 - 4\theta_{s,\text{off}}, 0.0389 - 2\Psi\right) - 3.7 l_s$	0.683
4	26	$\left(4.67 + 2\theta_{s,\text{off}}\right)\left(0.059 - 0.0191\alpha_r\arctan\left(0.0945, 2\Psi\right)\right)\left(\alpha_r - \alpha_s - 4\theta_{s,\text{off}}\right)$	0.654
5	21	$-0.18083\left(-2\theta_{s,\text{off}} + 2\Theta\left(-0.836 - 1.536\theta_{s,\text{off}}\right)\right) - \cos\left(0.778\alpha_r\right)$	0.640
5	19	$-0.156\left(-2\theta_{s,\text{off}} + 2\Theta\left(-0.971 - 2\theta_{s,\text{off}}\right)\right) - \cos\left(\arctan\left(\alpha_r, 0.521\right)\right)$	0.633
5	18	$-0.15576\left(-2\theta_{s,\text{off}} + 2\Theta\left(-0.971 - 2\theta_{s,\text{off}}\right)\right) - \cos\left(-0.796\alpha_r\right)$	0.633
6	28	$0.551 + 2\theta_{s,\text{off}}\left(0.723 - 0.586\alpha_r\right) + \sin\left(1.55 - 0.408\Psi - \alpha_r + e^{2\theta_{s,\text{off}}}\right)$	0.604
6	24	$0.81 + 1.7794\sin\left(2.74 - 0.26\Psi - \alpha_r + 2\theta_{s,\text{off}}\left(-0.561 + \alpha_r\right)\right)$	0.560
6	21	$-0.445\left(-1.31 + \alpha_r - 0.52\Psi\theta_{r,\text{off}}\right)\left(1.32 + \alpha_r + 2\theta_{s,\text{off}}\right)$	0.554

Table A.2: Additional equations for the six message elements of the 5 DOF manipulators. Extension to Tab. 8.11.