



Technisch-Naturwissenschaftliche
Fakultät

Symbolic Regression for Knowledge Discovery – Bloat, Overfitting, and Variable Interaction Networks

DISSERTATION

zur Erlangung des akademischen Grades

Doktor

im Doktoratsstudium der

Technischen Wissenschaften

Eingereicht von:

Dipl.-Ing. Gabriel Kronberger

Angefertigt am:

Institut für formale Modelle und Verifikation

Beurteilung:

Priv.-Doz. Dipl.-Ing. Dr. Michael Affenzeller (Betreuung)

Assoc.-Prof. Dipl.-Ing. Dr. Ulrich Bodenhofer

Linz, 12, 2010

For Gabi

Acknowledgments

This thesis would not have been possible without the support of a number of people.

First and foremost I want to thank my adviser Dr. Michael Affenzeller, who has not only become my mentor but also a close friend in the years since I first started my forays into the area of evolutionary computation.

Furthermore, I would like to express my gratitude to my second adviser Dr. Ulrich Bodenhofer for introducing me to the statistical elements of machine learning and encouraging me to critically scrutinize modeling results.

I am also thankful to Prof. Dr. Witold Jacak, dean of the School of Informatics, Communications and Media in Hagenberg of the Upper Austria University of Applied Sciences, for giving me the opportunity to work at the research center Hagenberg and allowing me to work on this thesis.

I thank all members of the research group for heuristic and evolutionary algorithms (HEAL). Andreas Beham, for implementing standard variants of heuristic algorithms in HeuristicLab. Michael Kommenda, for discussing extensions and improvements of symbolic regression for practical applications, some of which are described in this thesis. Dr. Stefan Wagner, for tirelessly improving HeuristicLab, which has been used as the software environment for all experiments presented in this thesis. Dr. Stephan Winkler, for introducing me to genetic programming and system identification and for proofreading a first draft of this thesis to improve its linguistic quality. Ciprian Zavoianu, for the interesting discussions about aspects of genetic programming.

I also wish to extend my thanks to my colleagues who invested a part of their valuable time to improve application-specific sections of this thesis. Christoph Feilmayr, for advice regarding data-based modeling of the blast furnace process and for proofreading and improving a draft of Chapter 8. Leonhard Schickmair, for his input on variable relevance metrics and for proofreading and improving a draft of Chapter 8. Dr. Stefan Fink, for preparing a dataset for economic modeling and for the discussion of economic models produced by genetic programming.

Finally, I thank my parents and my family who supported me and my interests in computer science. I thank all my friends for encouraging me to finish this thesis. This work is dedicated to Gabi for her love and support.

This thesis mainly reflects research work done within the Josef Ressel-center for heuristic optimization “Heureka!” at the Upper Austria University of Applied Sciences, Campus Hagenberg. The center “Heureka!” is supported within the program “Josef Ressel-Centers” by the Austrian Research Promotion Agency (FFG) on behalf of the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ).

Zusammenfassung

Mit der wachsenden Datenmenge, die in unterschiedlichsten Anwendungsbereichen gesammelt und aufgezeichnet wird, wächst auch das Bedürfnis, diese Daten sinnvoll zu nutzen. In der Wissenschaft haben Daten schon seit jeher einen hohen Stellenwert. In den jüngeren Jahren gewinnt aber auch das wirtschaftliche Potential von Daten zunehmend an Bedeutung. In Kombination mit Verfahren zur Datenanalyse kann dieses Potential ausgeschöpft werden, sei es im kommerziellen Bereich zur Optimierung von Angeboten, oder im industriellen Bereich zur Optimierung von Ressourceneinsatz oder Produktqualität basierend auf Prozessdaten.

In dieser Arbeit wird ein neuer Ansatz für die Analyse von Daten vorgestellt, der auf symbolischer Regression mit genetischer Programmierung basiert und das Ziel verfolgt, einen Gesamtüberblick über das Zusammenspiel von einzelnen Faktoren eines Systems zu ermöglichen. Dabei sollen möglichst alle potentiell interessanten Zusammenhänge, welche in einem Datensatz erkennbar sind, in Form von kompakten und verständlichen Modellen identifiziert werden.

Im ersten Teil der vorliegenden Arbeit wird dieser Ansatz der umfassenden symbolischen Regression im Detail beschrieben. Wesentliche Themen, die dabei eine Rolle spielen, sind die Vermeidung von Bloat und Überanpassung, die Vereinfachung von Modellen und die Identifikation von relevanten Einflussgrößen. In diesem Zusammenhang werden unterschiedliche Verfahren zur Vermeidung von Bloat vorgestellt und verglichen. Insbesondere wird der Einfluss von Nachkommenselektion auf Bloat analysiert. Darüber hinaus wird eine neue Möglichkeit zur Erkennung von Überanpassung vorgestellt. Im Zuge dessen werden darauf basierende Erweiterungen zur Reduktion von Überanpassung vorgestellt und verglichen. Eine wichtige Rolle spielt dabei das Pruning von Modellen, einerseits um Überanpassung zu verhindern und andererseits um komplexe Modelle zu vereinfachen.

Ein weiterer wesentlicher Aspekt ist die Analyse und Darstellung der umfangreichen Menge von unterschiedlichen Modellen, die aus dem vorgestellten Ansatz resultiert. In diesem Zusammenhang werden Möglichkeiten zur Quantifizierung von relevanten Einflussgrößen vorgestellt, die in weiterer Folge verwendet werden können, um Interaktionen von Variablen des analysierten Systems zu identifizieren. Durch die Visualisierung dieser Interaktionen entsteht ein Gesamtüberblick über das betrachtete System. Dies wäre alleine durch die Analyse einzelner Modelle, die sich auf spezielle Aspekte konzentrieren, nicht möglich. Zusätzlich wird im ersten Teil auch die Prognose von multivariaten Zeitserien mit genetischer Programmierung beschrieben.

Im zweiten Teil dieser Arbeit wird gezeigt, wie der vorgestellte Ansatz für die Analyse von realen Systemen eingesetzt werden kann und dadurch neue Einblicke ermöglicht werden können. Die Daten stammen von einem Hochofen für die Produktion von Stahl und von einem industriellen chemischen Prozess. Zusätzlich wird gezeigt wie derselbe Ansatz zur Identifikation von makroökonomischen Zusammenhängen eingesetzt werden kann.

Abstract

With the growing amount of data that are collected and recorded in various application areas the need to utilize these data is also growing. In science, data have always played an important role; in recent years, however, the economic potential of data has also become increasingly important. In combination with methods for data analysis, data can be utilized to their full potential, whether in the commercial sector to optimize offers, or in the industrial sector to optimize resources and product quality based on process data.

This work describes a new approach for the analysis of data which is based on symbolic regression with genetic programming and aims to generate an overall view of the interactions of various variables of a system. By this means, all potentially interesting relationships, which can be detected in a dataset, should be identified and represented as compact and understandable models.

In the first part of this work, this approach of comprehensive symbolic regression is described in detail. Important issues that play a role in the process are the prevention of bloat and over-fitting, the simplification of models, and the identification of relevant input variables. In this context, different methods for bloat control and prevention are presented and compared. In particular, the influence of offspring selection on bloat is analyzed. In addition, a new way to detect over-fitting is presented. On the basis of this, extensions for the reduction of over-fitting are presented and compared. Pruning of models is featured prominently, on the one hand to prevent over-fitting and on the other hand to simplify complex models.

An important aspect is the analysis of the vast amount of different models that results from the proposed approach. In this context, different methods to quantify relevant factors are proposed. These methods can be used to identify interactions of variables of the analyzed system. Visualizing such interactions provides a general overview of the system in question which would not be possible by analysis of individual models which are concentrated on selected aspects of the problem. Additionally, the prognosis of multivariate time series with genetic programming is described in the first part.

The second part of this work shows how the described approach can be applied to the analysis of real-world systems, and how the result of this data analysis process can result in the gain of new knowledge about the investigated system. The analyzed data stem from a blast furnace for the production of steel and an industrial chemical process. In addition the same approach is also applied on a data collection storing economic data in order to identify macro-economic interactions.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, December 15, 2010

Dipl.-Ing. Gabriel Kronberger

Contents

1. Introduction	7
1.1. Thesis Statement	10
1.2. Research Project Background	11
2. Machine Learning and Data Mining	13
2.1. Supervised Learning and Unsupervised Learning	13
2.2. Classification, Regression and Clustering	14
2.3. Reinforcement Learning	15
2.4. Time Series Forecasting	16
2.5. Generalization and Overfitting	16
2.5.1. Overfitting	17
2.5.2. Bad Generalization because of Incomplete Data	18
2.5.3. Time Series Modeling Pitfalls	18
3. Evolutionary Algorithms and Genetic Programming	21
3.1. Evolutionary Algorithms	21
3.2. Genetic Programming	22
3.2.1. Genetic Programming Variants	22
3.3. Bloat	24
3.3.1. Inviabile Code and Unoptimized Code	25
3.3.2. Bloat Control in Practice	25
3.3.3. Bloat Theory	26
3.3.4. Theoretically Motivated Bloat Control	27
3.3.5. Quantification of Bloat	28
3.4. Data-based modeling with Genetic Programming	29
3.4.1. Symbolic Regression	29
3.4.2. Overfitting	30
3.5. Offspring Selection	31
3.6. Genetic Programming with HeuristicLab	32
4. Interpretation and Simplification of Symbolic Regression Solutions	35
4.1. Bloat Control - Searching for Parsimonious Solutions	35
4.1.1. Static Depth and Length Limits	36
4.1.2. Parsimony Pressure	36
4.1.3. Dynamic Depth Limits	38
4.1.4. Experiments	41
4.1.5. Operator Equalization	42

4.1.6.	Pruning to Reduce Bloat	49
4.1.7.	Does Offspring Selection Reduce Bloat?	50
4.1.8.	Multi-objective GP	54
4.1.9.	Summary of Results of Bloat Experiments	54
4.1.10.	Effects of Bloat Control on Genetic Diversity	54
4.2.	Simplification of Symbolic Regression Solutions	57
4.2.1.	Restricted Function Sets	57
4.2.2.	Automatic Simplification of Symbolic Expressions	58
4.2.3.	Branch Impact Metrics	58
4.2.4.	Visual Support for Manual Simplification	58
5.	Generalization in Genetic Programming	61
5.1.	How to Detect Overfitting in GP	62
5.2.	Countermeasures Against Overfitting	64
5.2.1.	Restarts	64
5.2.2.	Parsimony Pressure Against Overfitting	65
5.2.3.	Pruning Against Overfitting	65
5.3.	Experiments	66
5.4.	Results	68
5.4.1.	Results: Covariant Parsimony Pressure	68
5.4.2.	Results: SGP and OSGP	70
5.4.3.	Results: Adaptive CPP and Overfitting-triggered Pruning	71
6.	Genetic Programming and Data Mining	77
6.1.	Genetic Programming	77
6.1.1.	Flexible Model Representation	77
6.1.2.	Implicit Feature Selection	77
6.1.3.	Non-Determinism	78
6.1.4.	Training Performance	79
6.1.5.	Predictive Accuracy	79
6.2.	Analysis of Relevant Variables	80
6.2.1.	Relation to Feature Selection	80
6.2.2.	Relative Variable Importance in Linear Models	81
6.2.3.	Variable Importance in GP	81
6.2.4.	Variable Importance in Random Forests	85
6.2.5.	Generalized Variable Importance	86
6.2.6.	Improved Formulations of Variable Importance Metrics for GP	87
6.2.7.	Validation of Variable Relevance Metrics	89
6.3.	Data Mining and Symbolic Regression	95
6.4.	GP-Based Search for Implicit Equations	95
6.5.	Comprehensive Search for Symbolic Regression Models	96
6.5.1.	Case Study: Chemical-II Dataset	96
6.6.	Improving GP Performance	102
6.6.1.	Parallelization	103

6.6.2. Improving Fitness Evaluation Performance	103
7. Multi-Variate Symbolic Regression and Time Series Prognosis	105
7.1. Evaluation of Multi-Variate Symbolic Regression Models	106
7.1.1. Curse of Dimensionality	108
7.2. Multi-variate Time Series Modeling and Prognosis	108
7.2.1. Evaluation of Multi-variate Time Series Prognosis Models	110
7.2.2. Case Study: Financial Time Series Prognosis	111
8. Applications and Case Studies	119
8.1. Blast Furnace	119
8.1.1. Modeling	121
8.1.2. Results	121
8.1.3. Result Details	124
8.1.4. Concluding Remarks	130
8.2. Econometric Modeling	131
8.2.1. Macro-Economic Dataset	131
8.2.2. Modeling	133
8.2.3. Results	134
8.2.4. Concluding Remarks	141
8.3. Chemical Process	144
8.3.1. Modeling	144
8.3.2. Results	145
8.3.3. Detailed Results	146
9. Concluding Remarks	151
A. Additional Material	155
A.1. Model Accuracy Metrics	155
A.1.1. Regression Models	155
A.1.2. Time Series Models	158
A.2. Numerical Approximation of Derivatives	161
A.3. Datasets Used in Experiments	161
A.3.1. Artificial benchmark datasets	162
A.3.2. Real World Datasets	163
A.4. Colophon	195
Bibliography	195
Curriculum Vitae	195

Overview

The main theme of this work is how genetic programming (GP) can be used for knowledge discovery. In particular, adaptations of GP to search for accurate, interesting, and understandable models are discussed. Additionally ways to filter and present the information gathered through GP runs are presented that aim to improve knowledge discovery process and lead to a better understanding of the examined system or process. This work puts strong emphasis on producing comprehensible models with GP, the accuracy of the output of the models is only secondary.

This work mainly concentrates on the practitioners view of GP and symbolic regression. Thus the majority of this work is about practical aspects, theoretic aspects are only treated briefly. GP theory has seen a lot of progress in the recent years. These contributions are very relevant for a better understanding of the internal dynamics of GP and provide a solid basis on which to conduct empirical or practically oriented research.

The main advantage of GP compared to other data-analysis methods is that it produces white-box models that might lead to new knowledge about the examined system. The price to pay for this is that GP also requires more computational resources than other data-analysis methods. The internal dynamics of genetic programming often leads to final solutions that are relatively complex and hard to understand. In the literature a large number of possible remedies for this issue have been described and analyzed, however, from a practitioners point of view the results are not satisfactory. In the first part the following novel concepts are presented:

- Comprehensive GP-based identification of interesting models or interactions in a given data-set.
- Data-based methods to quantify and visualize relevant interrelations of variables of a system.

The following issues relate to the main objectives and are also discussed in the first part:

- Methods to create compact and comprehensible models with GP.
- Methods to detect and avoid overfitting with GP.
- Symbolic vector-regression for the identification of multi-variate models.
- GP-based time series modeling for prognosis.

In the second part the concepts described in the first part are applied to a number of real world data mining problems.

1. Introduction

Well organized and accurate data have become increasingly important as a potential source of profits for businesses. As more and more data are generated and stored in large repositories the desire to generate value from the available data grows. Business intelligence as a discipline is intimately related to the collection, aggregation and analysis of business-relevant data. Decision support systems play an important role in business intelligence to analyze and summarize relevant data often in form of charts and condensed reports. The main purpose of decision support systems is to support managers to make correct profitable decisions. Decision support systems of various complexity are already ubiquitously implemented in many areas (politics, finance, health, industries...) and have an impact on our everyday life.

A large number of data-analysis methods for different applications have been described ranging from very simple methods to mathematically complex and powerful algorithms. The methods can be categorized in one or multiple of the following partially overlapping disciplines: System theory, statistics, machine learning, and data mining. All these disciplines have in common that they cover the topic of data-based modeling [124, 72, 27]. Additional research areas related to these disciplines are artificial neural networks and reinforcement learning [210]. *“Artificial neural networks are a development of the Perceptron [173] [...] and the area has transformed into a community of its own [...]. This is quite remarkable, since the topic is but a flexible way of parameterizing arbitrary hypersurfaces for regression and classification. The main reason for the interest is that these structures have proved to be very effective for solving a large number of nonlinear estimation problems.”* [124]. Recently a number of novel contributions have been published discussing learning for so-called deep networks with very many hidden layers for which back-propagation learning algorithms do not work well [85, 60, 132].

Data mining includes a wide range of different techniques to analyze, filter, explore and visualize data [79, 233]. Fayyad et al. define data mining in the following way: *“Data mining is the application of specific algorithms for extracting patterns from data”* [61]. Another definition is given by Hand et al.: *“Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.”* [79]. Based on these definitions many data-based modeling approaches can also be called data mining methods. Data mining also includes exploratory elements like drilling down into interesting facts in an interactive cyclic process of model construction and interpretation. Fayyad et al. define data mining as a step in the process of knowledge discovery from databases. The whole process also includes data selection, preprocessing and transformation as necessary preparatory steps for data mining and the interpretation and evaluation of the results obtained from data mining as a necessary follow up step to

achieve the stated goal of knowledge discovery. An essential requirement for effective data mining is that results are presented in a clear and informative way in order to allow a data miner to evaluate and interpret the results easily. Methods and principles for the visual display of quantitative data [217, 218] are closely related to data mining and knowledge discovery.

This work concentrates on the application of genetic programming for data mining of data collections that fit entirely into main memory. In particular, this work does not cover issues and solutions related to very large data collections that must be read sequentially from external memory. Genetic programming is a nature inspired meta-heuristic which is applicable to a wide range of problems in its general formulation. The idea of GP is to imitate aspects of biological evolution such as selection, recombination, and mutation for find computer programs that solve a given problem in small evolutionary steps starting with an initial population of random programs. Genetic programming in its most general formulation can be called an “*invention machine*” [104, 105]. In most cases, however, problem-specific variants of GP are used to solve well defined problems. One example for a specific and rather simple GP approach is symbolic regression; in this application GP is used to search for models that can be represented as symbolic expressions and approximate the response of a system based on observations of the input and output behavior of the system. Symbolic regression relies on the power of GP to find the correct structure for the model in combination with the model parameters. This property makes GP especially suitable for data-analysis tasks where only little information about the origin of the data and the examined system is available. Multiple authors [94, 232, 3, 231, 191, 126, 174, 18] have demonstrated that problem-specific GP variants for data-analysis can find very good solutions for specialized tasks like regression and classification. A more general approach is introduced in [154] which uses GP to generate classification algorithms which can later be executed to produce classifiers based on a specific data-set.

The general formulation of GP has the drawback that it often takes more computational effort to achieve solutions of comparable accuracy than it would take with other more specific methods. This disadvantage is often accepted as the time spent for model training is usually not critical. The time spent for training is often only a small part of the total time spent in the knowledge discovery process. A major part of the time is spent on data preparation and model interpretation and analysis [70]. If genetic programming is to be used to search for interesting patterns in large datasets it is beneficial if the training time can be reduced as more potentially interesting patterns can be uncovered in the same time frame. The chance to find previously unknown interrelations in the data grows with the number of good models. In Section 6.6 we describe methods to increase the training performance of GP that do not have a negative effect on the model accuracy.

Data analysis methods can be classified based the representation of models produced by the method. The spectrum ranges from comprehensible white-box models on the one end to complex black-box models on the other end. Often the distinction is not clear and various shades of gray-box models are located in between the two extremes. Models produced by symbolic regression are located near the white-box end of the spectrum.

Typically symbolic regression models are mathematical expressions which can be directly interpreted in order to gain knowledge about the patterns which were found in the dataset. Support vector machines are an example for an approach that produces black-box models located on the other end of the spectrum. Artificial neural network models are usually said to be gray-box models because the network structure can be interpreted but it is hard to gain a full understanding of the interrelations in the network.

While the fact that GP produces white-box models is often stated as an advantage this is often not apparent in practice. The models presented as final result is often rather complex and difficult to understand. Even though the model is a mathematical expression it is often impossible to gain an understanding of the model when it contains deeply nested functions. This problem is also a major topic in [94] which discusses the application of genetic programming for scientific discovery. One cause for this is that genetic programming has the tendency to build ever larger and more complex programs over time without a proportional improvement in program fitness. In the GP community this phenomenon is called code bloat and a lot of effort has been put into designing counter-measures that reduce or prevent this behavior. In Chapter 4 this phenomenon and a few selected counter-measures will be described in more detail. Even though a number of counter-measures has been described in the literature [188] there is not yet a consensus which methods are effective in practical applications. If symbolic regression is used for an unguided search for interesting patterns in a dataset then an effective anti-bloat strategy is very important because only small and at the same time accurate and generalizable models are interesting as a source for new insights into the nature of the dataset.

Another reason why it is important to create compact, comprehensible models is that the data miner is more likely to trust a model to be true when it is understandable and easy to validate. The issue of models which can be trusted should not be underestimated [99] as it is the first and foremost criterion that a data miner trusts the model before the model is examined in more detail. Demonstrating that a model can be trusted is even more important than the accuracy of the model. Models that are too accurate to be true might raise concerns that the model might be over trained and thus cannot be trusted (for a more detailed discussion of overfitting see Chapter 5). One way to increase the chance that a model is trusted is to reduce the complexity of the model, making it easier to validate and analyze the model. Another way is to demonstrate clearly that the model also is accurate on new data, for instance through validation on a hold-out set or through cross-validation. Even though such validation methods are common practice in the machine learning community these standards are not yet fully implemented in the evolutionary computation community [59]. In a recent contribution O'Neill et al. state that: “[...] *this issue in GP has not received the attention it deserves and only few papers dealing with the problem of generalization have appeared [...]*” [148].

In the original formulation of symbolic regression with genetic programming a single variable must be explicitly declared as the target variable [101]. However, often it is not immediately clear which variable is the variable of interest or there are multiple variables that could be declared as a target variable. So it is often necessary to explore the options with multiple symbolic regressions runs with different configurations.

Another data mining task that is relevant in practice but has not yet been treated extensively in the GP literature, is multi-variate symbolic regression. In this approach GP has to find a symbolic multi-variate regression model that approximates all target variables. If GP is to be used for unguided data mining without an explicit target variable the solution representation and the search process of genetic programming have to be extended. In Chapter 6 a simple approach of independent runs to create and collect models for all variables of the dataset is described. In Chapter 7 several extensions of the solution representation for the simultaneous modeling of multiple variables are described.

1.1. Thesis Statement

To summarize the previous paragraphs genetic programming can be used for data mining to find potentially interesting patterns or interactions in large datasets when the traditional symbolic regression approach is extended and adapted appropriately.

- Solution representation: Instead of a single target variable the process should identify the potentially interesting variables automatically. This work describes several different ways to extend the solution representation for data mining tasks and introduces ways to manage the potentially large set identified models.
- Model complexity and presentation: In order to facilitate knowledge discovery from GP models, appropriate and effective methods for reduction of model complexity have to be integrated into the search process. This work describes different methods to reduce the model complexity and introduces new ways of visualizing results to simplify detailed analysis of models.
- Trustable models: Results presented by the process must be clearly validated to enable the data miner to estimate how trustable a model is. This work describes a validation procedure for GP that is effective in practical applications and describes how visualizations can be used to improve the trustability of models.
- Performance: In order to make unguided search for interesting patterns in large data sets feasible it is necessary to reduce the training time of GP. In this work briefly discusses possible ways to increase GP performance.

Chapter 2 gives a cursory introduction of machine learning and data mining including commonly used terms. Chapter 3 discusses relevant previous work in the area of evolutionary algorithms in general and genetic programming specifically. Chapter 4 is the first core chapter of this thesis and discusses the problem of bloat and countermeasures to reduce bloat and also describes ways to simplify solutions with pruning. Chapter 5 introduces an effective way to detect overfitting and countermeasures against overfitting in GP. Chapter 6 describes unconstrained GP-based search for interesting models and how the information collected in those experiments can be prepared and visualized to improve the knowledge acquisition process. In Chapter 6 the topic of variable relevance

metrics is discussed as it is necessary for the representation of variable relation networks. Chapter 7 describes an approach to create multi-variate symbolic regression models with GP and how multi-variate auto-regressive time series models can be used for the prognosis of future values of a multi-variate time series. Finally, Chapter 8 demonstrates how the methods discussed in the previous chapters can be applied to real world applications.

1.2. Research Project Background

This thesis mainly reflects research work done within the Josef Ressel-center for heuristic optimization “Heureka!” at the Upper Austria University of Applied Sciences, Campus Hagenberg. The center “Heureka!” is supported within the program “Josef Ressel-Centers” by the Austrian Research Promotion Agency (FFG) on behalf of the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ).

2. Machine Learning and Data Mining

Essentially, all models are wrong, but some are useful.

George Box

This chapter gives a brief overview of the disciplines of machine learning and data mining. A number of terms frequently in the machine learning literature that are necessary to understand the core of the work are explained in the following sections. The reader who is already familiar with machine learning can skip this chapter and advance to the chapter.

Learning is the process of knowledge acquisition from observations. Knowledge is defined by the Oxford Dictionary of English as: “(i) facts, information, and skills acquired through experience or education; the theoretical or practical understanding of a subject; (ii) the sum of what is known; (iii) information held on a computer system.” [152]. One of the goals of machine learning as a research discipline is to define algorithms which when executed by a computer enable it to learn general facts from examples in such a way that previously unseen examples can also be recognized and a correct action is triggered.

As such the knowledge learned by the machine learning method should be specific to recognize known examples and general to also recognize slightly different new examples. In statistics the learning or training phase is called model estimation. Model estimation is the process of selecting a specific model from a class of models that fits the available data best.

Typically a discrimination is made between the two terms features and variables. In machine learning the term variable usually indicates the original variable while a feature can be a combination of multiple variables. Often kernel functions are used to transform a combination of variables to features. In this work no distinction between the two terms is made and the terms variable and feature are used interchangeably.

2.1. Supervised Learning and Unsupervised Learning

Learning methods can be categorized into two general classes: supervised and unsupervised methods. The methods have different goals and are also different by the way how examples are presented in the training or learning phase.

Supervised learning methods accept pairs of training examples with a correct label as input. From these training examples the algorithm should learn how to correctly label

new examples in a generalizable way and learn from these examples which label fits best for future unseen examples.

Unsupervised learning methods accept training examples without labeling future input examples are compared to the previously learned training examples and one or multiple samples, that are most similar to the current example, are produced as output.

The semi-supervised learning hybrid approach is also possible. This is often useful when a large number of examples are available but only a small part of them is labeled. Semi-supervised learning methods determine labels for the unlabeled examples by finding similar labeled representatives and then use the unlabeled examples for training. Typical tasks solved by supervised learning methods are classification and regression; unsupervised learning is often used for clustering.

2.2. Classification, Regression and Clustering

The input data for classification, regression and clustering are usually a collection of n training examples. Each example in the training data is a row vector of k values. The set of training examples can be represented as a matrix $T_{(n,k)}$.

For the classification task the training samples have class values from a finite usually small domain. The most common case is binary classification, meaning that the class labels can have only two values (e.g. 0 or 1, malign or benign). However classification for three or more distinct class labels is also possible. It is not necessary that an ordering relation can be defined on the class labels. The goal of classification algorithms is to learn from the presented training samples in which class to categorize new and previously unseen data points. The quality of classification algorithms is measured by a loss function. Usually the loss function compares the class predicted by the algorithm with the actual class of the data point and then calculated from the number of correctly and incorrectly classified data points. Examples for classification algorithms are: linear discriminant analysis, C4.5 (decision tree learning) [169], support vector machines (SVM) [222, 41], k-nearest-neighbor (kNN) [42, 65], classification and regression trees (CART) [28].

For the regression task the training samples usually in \mathbb{R}^n are labeled with values in \mathbb{R} . The algorithm should learn the value linked with each data point so that it can generate a predicted value for new and previously unseen data points. Labels can be ordered so the loss of a regression model is often measured as a distance of the predicted from the actual target values. Examples for regression algorithms are: linear regression, CART [28], SVM regression [221].

The task of binary classification can be reformulated into the regression task of finding a discriminant function $g : \mathbb{R}^n \rightarrow \mathbb{R}$. The classification is achieved by assuming a constant separator value c so that class = 0 if $g < c$ and class = 1 if $g \geq c$. A perfect prediction can be achieved when a value c exists that separates both classes perfectly.

Regressing and classification are supervised learning tasks. In contrast, clustering is an unsupervised learning task [79, 83]. The goal of clustering is to group examples by similarity into a number of representative clusters. Each cluster is a set of examples and

a representative example. New examples are assigned to one of the learned clusters into which the new examples fits best. Clustering can also be applied to unlabeled examples.

A typical example of clustering is market basket analysis based recommender systems. The system uses a clustering method to find clusters of customers who purchased similar products. Based on the clustering the recommender system can suggest related products based on the products that other customers in the same cluster frequently purchased. Examples for clustering algorithms are: k-Means [125], expectation-maximization (EM) [49], and methods based on principal component analysis (PCA) [156, 89].

2.3. Reinforcement Learning

Reinforcement learning is a branch of machine learning that does not fit into the categories supervised vs. unsupervised learning. Reinforcement learning is concerned with online processes in which an agent can take actions and receives rewards for actions [210]. The acquisition of new observations which implicitly cause an unknown loss is integrated directly in the learning process. The objective is to find a policy for the agent that minimizes the regret, which is the difference of the reward that the agent received to the reward that the agent could have received, if it would have executed the optimal actions at all times. The difference to supervised learning is that the correct actions are never explicitly presented to the agent.

The best example to illustrate reinforcement learning methods is the multi-armed bandit problem that describes the situation of a rigged casino, a bandit gambling machine with multiple arms with a different rate of winning plays and possibly unequal rewards. The goal for a player is to allocate plays to the machines in such a way, as to maximize the expected total reward. The probability distribution of rewards for each arm is unknown, so an effective strategy has to be found that balances exploration of new arms and exploitation of the best arm so far. The K -armed bandit problem can be described as K random variables with $X_i (0 \leq i < K)$, where X_i is the stochastic reward given by the arm with index i . The rewards X_i are independent and the distributions are generally not identical. The laws of the distributions and the expected value μ_i for the rewards X_i are unknown. The goal is to find an allocation strategy that determines the next arm to play, based on past plays and received rewards, that maximizes the expected total reward for the player. Or put in other words: to find a way of playing that minimizes the regret, which is defined as the expected loss occurred by not playing the optimal arm each time.

Since a player does not know which of the arms is the best he can only make assumptions based on previous plays and received rewards and concentrate his efforts on the arm that gave the highest rewards so far. However it is necessary to strike a good balance between exploiting the currently best arm and exploring the other arms to make sure that an arm with a higher expected reward value can be discovered.

2.4. Time Series Forecasting

The notable difference of time series forecasting in comparison to regression or classification is that there is a time-dependent structure in the processed data points. Special care has to be taken in the training phase and when using the trained model so that the time-dependency or the ordering of the data points is not destroyed. In particular it is not valid to shuffle the data points in the training phase as this would destroy the internal structure.

The goal of time series forecasting is to predict future values of a variable based on past values of the variable and optionally other related variables. If the method uses past values of the target variable for the prediction of future values it is an auto-regressive method.

Time series forecasting is a traditional application of statistics. Since the very first formulations of time series models a large corpus of ever refined modeling approaches has been described ranging from simple linear auto-regressive and moving average models [74, 90] to intricate models including trends and seasonal changes and multivariate time series [229]. For an extensive discussion of statistic approaches to time series modeling see [23, 29, 30, 157, 57, 82].

Genetic programming has also been effectively used for time series prognosis tasks [101]. The power of genetic programming to evolve the structure of solution candidates in combination with the parameters enables the process to automatically adjust the model structure appropriately for instance to account for seasonal changes, trends or recurrent events if such elements are detected in the data.

Chapter 7 discusses multi-variate time series modeling with genetic programming.

2.5. Generalization and Overfitting

Any data-based model, regardless of which learning or estimation process is used, must generalize well in order to be useful. Models which generalize well are models that also work consistently for new observations. In contrast models that do not generalize well produce inaccurate or even largely incorrect estimations for new observations and are thus useless in practice. The generalization ability of a given model can be estimated by using a hold out dataset which contains samples that are not used in the training process. The fit on the hold-out is an indicator for the fit on new observations. If the model performs equally well on the training set and on the hold-out set it can be assumed that the model generalizes well. This assumption only holds in general when a number of assumptions about the analyzed system and the selection the training set and the hold-out set are fulfilled. The number of different states of the system must be limited and the training set and the hold-out set must be a representative sample of the whole population of observable states of the analyzed system. If these assumptions hold new observations from the same system will be similar to observations in the training- and hold-out set, and the fit on the hold-out set is an accurate estimator for the expected fit for new observations.

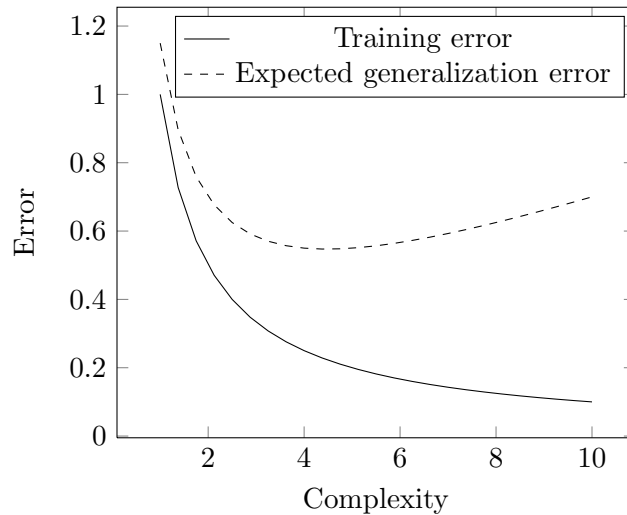


Figure 2.1.: Underfitting and overfitting

First we discuss possible causes for bad generalization behavior in an ideal setting where all the assumptions hold. In practice the situation is more difficult as some of the assumptions might not hold. This will be discussed later in more detail.

2.5.1. Overfitting

Bad generalization ability is often the result of overfitting. Overfitting occurs when the model class from which a model is selected is too complex. From a complex model class a model can be selected that fits any point in the training set very closely thus the model class has a low bias. If one training point is left out or an additional point is added this leads to a very different model. Even though the new model still fits all points of the training set very closely it has a very different overall shape thus a model class with large complexity has a large variance. In contrast under-fitting occurs when the model class is too simple. A simple model class has a large bias because the model cannot fit every point closely but low variance because if any point in the training set is changed the resulting model is not very different. This is the well known bias-variance trade-off of statistics. Minimizing the bias to get a better fitting model on the training set inevitable leads to higher variance and worse generalization. Thus the model class has to be selected in a way to find a good balance. Figure 2.5.1 shows how the training error is reduced by increasing the model class complexity while the expected model error increases. Under-fitting occurs on the left where the training error and the expected model error are large. At a certain complexity the expected model error is minimal at the optimal balance between bias and variance. On the right side overfitting occurs and the expected model error grows with increasing model class complexity.

In summary, overfitting can be prevented by restricting the complexity of the model class. Often this is done manually through model validation on a hold-out set or through

cross-validation. Such methods can be used with any kind of data-based modeling algorithm. Some algorithms include countermeasures for overfitting into the learning process. Often this is done by adding a complexity term to the objective function so that the model accuracy is optimized together in combination with the model complexity as for instance used in SVMs. Another approach is to use an internal validation set to control the complexity of models as for instance used in enhanced implementations of ANN.

2.5.2. Bad Generalization because of Incomplete Data

In real world scenarios bad generalization can often be the result of incomplete data. The assumption that the training set is a set of independently sampled observations and representative for the whole population of possible observations does often not hold.

If the training set does not contain examples for all system states that are likely to be observed, it is not possible to create a model that will generalize well to all new observations. The learning algorithm can only build a model based on the incomplete information in the training set. If the state of the analyzed system or its inputs are changed the system behavior will also change and the model does not fit anymore. A model trained from an incomplete training set will not generalize well if the behavior of the underlying system changes. It should be noted that this problem is independent of overfitting. Even if the model showed good fit via cross-validation the generalization error can be large as cross-validation assumes that the samples in the training set and hold out were sampled independently. Even though the bad generalization is in this case not caused by overfitting in the original definition but by incomplete data the remedies to improve the generalization are similar. Reduction of complexity to increase the bias and reduce the variance of the model class should lead to better generalization behavior in both situations. In the remainder of this work we will thus refer to both causes for bad generalization as overfitting.

2.5.3. Time Series Modeling Pitfalls

Time series have an implicit time-dependent structure which must be preserved for modeling and estimation. In particular two observations of a time series are not independent. It should be noted that many collections of observations are by definition a time series as the observations are usually made over a certain timespan and not simultaneously. Often the original time-dependency is dropped, however, when it can be argued that the observations are independent. In order to drop the time-dependency it must be shown that all pairs of observations $x(i)$ and $x(j)$ ($i > j$) are independent (nothing that effected observation $x(i)$ can have an effect on observation $x(j)$). In particular $x(t+1)$ must be independent from $x(t)$. A simple but non-sufficient criteria to check is if $x(t)$ and $x(t+k)$ are correlated. Alternatively visual examination is also a quick way to determine if the data has a time-dependent structure.

In many situations the time-dependency must be preserved. Examples are series of measurements of technical systems, financial or economic time series, series of diagnostic values of patients over time.

As a consequence shuffling of examples is not allowed for time series as it would break up the time-dependency. Shuffling of examples is often done as preparatory step before training or cross-validation, and is implemented in this way in many algorithm implementations (e.g. libSVM [32]). For time series the examples in the training- and hold-out set must be a set of consecutive observations. Semantically it is more appropriate to use samples $x(0)$ to $x(k)$ as training set and samples $x(k+i)$ $i \in [1..n-k]$ as hold-out set, forecasting from previous observations.

3. Evolutionary Algorithms and Genetic Programming

3.1. Evolutionary Algorithms

Evolutionary algorithms imitate aspects of biological evolution such as selection, recombination, and mutation to find a solution to a given problem starting with an initial population of random solution candidates. From the initial population two solution candidates are selected as parents and recombined to produce one or two offspring individuals which are then optionally mutated and subsequently added to the population. Parent selection has a higher chance to select solution candidates with an above average fitness and recombination has the effect to combine the traits of the parent individuals in the offspring individuals. This is related to the principle of “survival of the fittest” in biological evolution. The selective bias and the combination of positive traits in individuals have the effect, that the average fitness of the population increases over many generations producing better and better solution candidates over time.

The pioneers of evolutionary algorithms are Fogel who first described evolutionary programming [67], Rechenberg who first described evolution strategies [171], and Holland who first described genetic algorithms [86] at around the same time. The three algorithms all imitate aspects of biological evolution, however, each of the authors used a slightly different approach. The first descriptions of genetic algorithms use a binary solution encoding and emphasize the aspect of parental selection and sexual reproduction and recombination of positive traits. In contrast the first descriptions of evolution strategies use a real-valued encoding and emphasize the aspect of mutation and selection for survival of excess offspring. Self-adaptive aspects also play a large role in evolution strategies and have been integrated to improve the algorithm already very early [185]. In the first description of evolutionary programming graphical models in the form of finite state machines for the prediction of a series of symbols are evolved through a combination of sexual reproduction with crossover and mutation. Evolutionary programming also emphasizes the selection of offspring with above average fitness [66] and has thus been compared mainly with evolution strategies.

The fundamentally different approaches of genetic algorithms, evolution strategies, and evolutionary programming initially lead to a fragmentation of the research communities in separate camps. However, this initial fragmentation became gradually less distinct over the years, as the approaches have been further developed and aspects of genetic algorithms have also been integrated into evolution strategies, and vice versa.

3.2. Genetic Programming

Genetic programming [101] is an evolutionary algorithm to find computer programs that solve a given problem when executed. In contrast to genetic algorithms and evolution strategies which directly evolve solution candidates for the problem, the individuals in GP are computer programs which can be interpreted or executed to produce a solution to the original problem. The structure of GP solution candidates is not fixed, in particular, the length of programs evolved in GP is not predetermined. This is a major difference to other evolutionary algorithms which use fixed-length solution encodings for instance real-valued vectors, bit-strings, or permutation arrays.

Solution candidates in GP are most often encoded as symbolic expression trees that represent computer programs. The set of allowed symbols in a tree and the evaluation of trees are problem specific, and can be adjusted through algorithm parameters. These parameters are the *function set*, *terminal set*, and the *fitness function*. The function set contains symbols that can be used as internal nodes of the tree and the terminal set contains symbols which are allowed as terminal nodes.

Generally genetic programming can be used to evolve solutions for a given problem, if it is possible to define a language describing possible solutions and a fitness function for such solutions. Genetic programming has been used to find novel and human competitive solutions to hard problems and to find re-discover previously patented solutions [103, 14, 19, 105, 6, 133, 102].

3.2.1. Genetic Programming Variants

Many different variants of genetic programming have been studied in the literature and there is no precise definition of a genetic programming algorithm. Instead the term genetic programming encompasses all the different algorithm variants. As stated by Poli et al., “*At the most abstract level GP is a systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done.*”.

The different variants of GP can be differentiated into different classes by the way how the programs are represented (solution encoding). Koza-style GP [101], often called standard GP, uses a tree-based solution encoding. This representation of programs has the advantage that recombination and manipulation operators are relatively easy to implement. One problem of Koza-style GP is the requirement that the function set must have the closure property, which means that the types of all terminals, function results, and function arguments must be compatible. This property is necessary to guarantee that all possible tree shapes can be evaluated. The only syntactic constraint available in standard GP is the number of sub-trees allowed for each function symbol. This simple assumptions of standard GP make it possible to define effective and easy to implement recombination and mutation operators for the evolution of trees. More complex GP variants are often more powerful but also significantly more difficult to implement correctly, especially because the evolutionary operators are often more complex.

In some applications it is necessary that the possible structure and characteristic of GP

solutions can be defined more precisely. This issue has been addressed in two different but related ways, leading to the definition of grammar-based GP and strongly typed GP.

Grammar-based GP

In grammar-based GP [228, 234] all possible GP solutions are defined through a grammar for GP solutions. The grammar defines the syntax of the programming language used by GP to express solutions, in the same way as the grammar of a programming language defines the possible programs that can be written in this language. In grammar-based GP it is easy to restrict the structure of GP solutions. This is for instance necessary if GP is used to evolve programs for a given existing programming language. Grammar-based GP has for instance been used successfully to evolve classification algorithms represented as Java programs [154]. A successful variant using this approach is grammatical evolution [147]. In grammar-based GP the grammar is used only to restrict the possible shapes of solutions. In grammatical evolution the grammar is also used to construct trees and from a linear representation. The solution is encoded as a variable-length list of integers. Each element defines which branch in a syntax rule must be taken to construct a valid tree. The translation process starts with the first element of the list and the main entry point of the grammar.

Strongly-typed GP

In strongly typed GP [136] the types of terminals, functions, and function arguments are declared. Each time the GP system produces a new tree for instance through crossover, it must make sure that the type of the argument is compatible to the type of the parameter of the function. As long as the number of possible types handled by the GP system is rather small the type constraints can be handled also through syntactical constructs using grammar-based GP, however, a strongly-typed GP system is more appropriate if the GP system must be capable to handle many different types.

Strongly-typed GP and grammar-based GP define constraints for the construction of trees, thus in the implementation of such GP systems the operators for initialization, recombination, and mutation must be extended to check all constraints. Both variants are generalizations of standard GP, which is essentially a strongly typed GP system with only one type instance and a very simple grammar that specifies only the number of arguments of functions.

Linear GP

In linear GP [21] the program code is evolved directly in a variable-length linear representation instead of a tree-based representation. The solutions represented in linear form are closer to the executing machine than the tree-based programs and thus evaluation of solution candidates is usually more efficient in linear GP than in tree-based GP variants. In the most extreme case linear GP directly evolves machine code [145]. Because linear code lacks the structure available in tree-based representations memory operations to

read and write stores or registers are necessary. In a recent contribution a linear GP approach to evolve Java byte-code is described [150].

PushGP [198] is an especially interesting strongly-typed linear GP system which uses a stack-oriented execution model. In the execution of *Push* expressions a separate evaluation stack is used for each occurring type and it is allowed to push quoted code-fragments. Thereby, it is possible to manipulate and transform the program code before execution. It has been suggested that PushGP can be used as an “autoconstructive evolution system” [198, 197], where the evolutionary operators used to recombine and manipulate operators are evolved simultaneously with the solutions. This is a very powerful and general approach which allows the GP system to self-adapt to the specific problem that must be solved.

Graph-based GP

In graph-based GP solution candidates are represented in form of a graph which is a more general data-structure than a tree. Graph-based programs can potentially be executed in parallel fashion [159]. The drawback of the approach is that the implementation of crossover and mutation operators for the evolution of graphs is more complex. Additionally, the interpretation of solutions is difficult.

Cartesian GP [81, 135] is a special form of graph-based GP. The execution model also uses a graph-based program representation, but in Cartesian GP a level of indirection is introduced as the graph is encoded in linear form. The evolutionary operators are defined on the variable-length linear representation which is translated into a graph data-structure for evaluation. The solutions are encoded as integer tuples each one describing the properties of a cell in a two-dimensional grid. The elements in the tuples define the symbol of the node, and the incoming edges from other cells. Cartesian GP was initially introduced for the evolution of electronic circuits [134]. Recently extensions to Cartesian GP have been described to allow the definition and execution of self modifying programs [81]

3.3. Bloat

The term bloat in genetic programming relates to the growth of program length over a run without a proportional improvement in fitness [117]. The effect has also been described as “*survival of the fattest*” [58]. Genetic programming uses a variable-length solution encoding so average program length of the solution candidates in the population can grow or shrink over a GP run. The randomly generated programs in the first generation are limited to a certain range of possible sizes. Changes in average program size are thus a result of the evolutionary operators acting on the population. It has been observed already early that standard GP bloats without special countermeasures to prevent unlimited growth of programs. Bloat is problematic because of the memory consumption and the increased computational effort necessary to evaluate bloated solutions. Additionally, bloated solutions produced by GP are difficult to understand and validate. Thus the topic has been studied intensively in the literature and a number

of different hypotheses for the cause of bloat and methods to control bloat have been described.

3.3.1. Inviability Code and Unoptimized Code

Bloated programs contain a lot of code that has no or only minor effect on the output of the program. The ineffective code fragments can be differentiated into two classes, namely inviable code and unoptimized code [16, 188, 238]. Inviability code fragments (introns [17]) are not executed at all and so have no effect on the program behavior. Introns can occur when non-sequential execution flow for instance conditional evaluation of code fragments is possible in the GP solutions. It is relatively easy to detect inviable code through dynamic code analysis. All code fragments that are not visited when the program is executed are inviable code and can be removed easily without altering the program behavior.

Unoptimized code fragments, in comparison, are executed and have an effect on the program output. However, the code fragment is either detrimental to the fitness of the program, or its contribution to fitness is not proportional to its length. In general unoptimized code is executed but can be replaced by more compact code or even removed completely in the case of detrimental code fragments. It is rather difficult to detect or replace unoptimized code. One approach that can be used to partially remove unoptimized code fragments for tree-based GP is pruning [232]. Pruning determines unoptimized branches by calculating for each branch of the original program the fitness of a transformed program where that branch has been replaced by neutral code. If the fitness of the transformed program is not significantly worse than the fitness of the original program the branch is unoptimized code and can be removed. In Chapters 4 and 5 we discuss pruning in more detail.

It has been suggested that symbolic regression is not prone to the propagation of inviable code, but very much affected by unoptimized code [186, 187]. This is however still an open question [129].

3.3.2. Bloat Control in Practice

Out of the necessity to control bloat a number of effective methods have been proposed for practical applications. A straight forward approach is to add static size and depth limits for trees [101]. The crossover and mutation operators check if newly created trees are within the size limits and invalid trees are discarded. Another approach are size-fair evolutionary operators that produce a new tree that has the same size as the original tree [112, 165] or mutation operators that actively reduce the program length [96, 15]. Alternatively, selection can be adjusted to control bloat. Parsimony pressure increases the probability to select smaller individuals by including a weighted penalty term that depends on program length into the fitness function [239]. Lexicographic tournament selection is an extension of tournament selection where the smaller solution is selected if two solutions have the same fitness value [128].

3.3.3. Bloat Theory

Even though the bloating effect of GP has been observed already early, the cause for bloat has long remained an open question. Over the time different theories for the cause of bloat have been formulated often in combination with methods to control bloat. In the following a number of frequently cited hypothesis for bloat are discussed, as the topic is also very relevant for this thesis. A good survey of past and current bloat theories is given in [188]. The topic will be revisited in Chapter 4.

Defense against crossover

An early theory for bloat was that bloated solutions are more robust against disruptive changes by crossover [10]. The probability that crossover has a strong negative effect on the fitness of an individual is smaller for solutions that bloated and contain many introns relative to compact solutions. This means bloated individuals are preferred by selection as they are not destructed by crossover so easily and over time the evolutionary process leads to more bloated individuals in the population. The theory has been disputed [188] and is currently not considered to be a valid theory for the cause of bloat in tree-based GP.

Removal Bias

The removal bias theory of bloat [195] is closely related to the defense against crossover theory and also relates the cause for bloat to the fact that a surplus of inviable code makes it easier to add more code without negative effects on fitness. In particular, fitness is not effected strongly when crossover affects an inviable branch. In such crossover events there is an asymmetry, namely that the sub-branch which is removed from the inviable branch has a limited size, but the branch that is inserted from the other parent can be of any size and in particular larger than the removed branch. This asymmetry can lead to code growth even when the crossover is protected, producing only offspring that are strictly better than their parents. The removal bias theory has also been superseded by the more recent crossover bias theory.

Fitness causes bloat

The fitness causes bloat theory relates the cause for bloat to the nature of the program search space [111, 116]. In this theory introns are merely an effect of bloat but not a cause, in contrast to removed bias theory or defense against crossover theory. Fitness causes bloat theory proceeds from the assumption that bloat does not occur when no selection pressure is applied to the population. It has been shown that bloat does not occur when a constant or random fitness function is used [20, 116, 110]. If crossover is applied repeatedly on a number of solutions with only random selection (or with flat fitness) no code growth can be observed.

The theory states that bloat occurs if the program search space has the property that any solution can be expressed in many alternative but semantically equivalent ways,

and that there are more longer than shorter alternative representations. An additional requirement for the occurrence of bloat is a static fitness function that assigns the same fitness to all semantically equivalent programs regardless of their size. The length distribution of equivalent programs causes a drift to larger solutions.

The theory is general and is applicable to all iterative search algorithms with a discrete variable-length solution encoding and a static fitness function. In particular, bloat is not specific to population-based algorithms but can also occur in trajectory-based algorithms and in algorithms that do not use a crossover operator.

Crossover bias

Crossover bias theory [164, 52] is related to fitness causes bloat theory and is the most recent theory for the cause of bloat. It states that bloat occurs because of a bias of the combination of sub-tree crossover and selection. As stated in the fitness causes bloat theory bloat only occurs in the presence of selection pressure and a program search space where for a given fitness many more larger solutions than small solutions exist.

Crossover bias theory is based on the observation that sub-tree swapping crossover in tree-based genetic programming produces offspring with a particular size distribution (Lagrange distribution of the second kind). The average size of offspring is not altered by sub-tree swapping crossover, as the removed branch and the inserted branch have the same size on average. However, crossover produces more smaller individuals than large individuals. It has been shown that the size distribution of individuals produced by crossover depends on the symbols in the function set and on the number of parameters of the functions [55].

The combined effect of this crossover bias in combination with a fitness landscape where smaller individuals are more likely to have lower fitness than larger individuals causes a drift to larger individuals. Actually, any operator that produces a surplus of smaller individuals leads to bloat. Thus, it has been suggested recently to rename the theory to *operator length bias* theory [55].

3.3.4. Theoretically Motivated Bloat Control

A number of effective bloat control methods are based on crossover bias theory. As bloat control is a major topic in this thesis these methods are discussed briefly in the following. In Chapters 4 and 5 bloat control methods are discussed in more detail.

Tarpeian Method

Tarpeian bloat control sets the fitness of individuals with above average length to a very low value which effectively zeros the chance of selection of this individual. This method of bloat control is very simple and can be controlled through only one parameter that determines the chance that a given individual is assigned a zero fitness. The effect of Tarpeian bloat control is that the dynamic holes in the fitness landscape reduce the chance of selecting larger individuals and bloat is prevented or at least reduced. Recently, the covariant Tarpeian method for bloat control has been introduced [161], where the

probability that a solutions fitness is zeroed, is automatically adapted based on the covariance of program lengths and fitnesses in the population.

Covariant Parsimony Pressure

The covariant parsimony pressure method to control bloat [163] is based on the size evolution equation [162]. The parsimony pressure method adjusts selection probability to prefer smaller individuals relative to larger individuals with the same fitness. For this an adjusted fitness value is calculated that includes not only the raw fitness but adds a penalty term depending on the length of the solution. In covariant parsimony pressure the penalty term is adjusted dynamically based on the covariance of fitness and length over all individuals in the population. It has been shown that the average program length can be controlled tightly using covariant parsimony pressure to completely remove bloat.

Operator Equalization

The operator equalization method to control bloat [54] is also based on crossover bias theory and removes bloat by adjusting the distribution of individuals produced by the evolutionary operators. This is accomplished by an equalization step which filters newly created individuals, so that the length distribution of solutions in the next population matches a specific target distribution. As soon as the frequency of individuals of a specific size matches the target frequency, newly created individuals of the same size are discarded. The method continues creating new offspring using the evolutionary operators until the population can be fully filled and the size distribution in the population matches the specified target distribution. A self-adaptive variant of operator equalization has also been described, where the target size distribution is adjusted to follow the fitness distribution while still preventing bloat [189, 190]. In comparison to Tarpeian bloat control which hooks into fitness evaluation to control bloat, operator equalization removes the other necessary ingredient for bloat, namely the operator length bias.

3.3.5. Quantification of Bloat

In a recent contribution a measure for bloat has been defined that can be used to compare the relative amount of bloat in GP runs [220]. Bloat is the disproportional growth of program length relative to fitness improvement, thus the function shown in Equation 3.1 measures the amount of bloat at generation g as the relative change of average program length $\bar{\delta}$ from the initial generation to generation g over the relative change of average fitness \bar{f} in the same interval.

$$\text{bloat}(g) = \frac{(\bar{\delta}(g) - \bar{\delta}(0))/\bar{\delta}(0)}{(\bar{f}(0) - \bar{f}(g))/\bar{f}(0)} \quad (3.1)$$

If no bloat has occurred the function $\text{bloat}(g)$ has a value of one, indicating that the average program size and the average fitness changed by the same amount. Notably, the function can also become negative if the average program length decreases while the average fitness increases.

The original definition shown in Equation 3.1 only works for minimization problems, for maximization problems the relative change in fitness can be calculated as $(\bar{f}(g) - \bar{f}(0))/\bar{f}(0)$. A problem occurs when $\bar{f}(0)$ is close to zero which leads to unstable results. This can happen for instance when the squared correlation coefficient is used as fitness measure. To mitigate such problems the calculation of the relative change in fitness should be adjusted accordingly as shown in Equation 3.2.

$$\text{bloat}(g)^{max} = \frac{(\bar{\delta}(g) - \bar{\delta}(0))/\bar{\delta}(0)}{(\bar{f}(g) - \bar{f}(0))/(1 + \bar{f}(0))} \quad (3.2)$$

In Chapter 4 this function is used to compare the effect of different bloat control methods.

3.4. Data-based modeling with Genetic Programming

3.4.1. Symbolic Regression

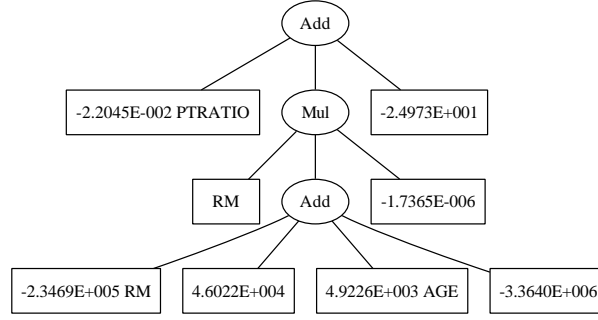
One possible problem that can be solved by genetic programming is symbolic regression. Symbolic regression [101] is concerned about finding a model $\hat{f}(x)$ (functional expression), that is a good approximation of the values of the target variable y given a number of input variables x . The values of the target variable are produced by an unknown response function of the studied system $f(x)$. The input for symbolic regression is a dataset with n observed values of each variable of the studied system. The result is a functional expression $\hat{f}(x)$ encoded as a symbolic expression tree. For symbolic regression the GP function set usually includes at the arithmetic operators $(+, -, *, /)$, and the terminal set includes all allowed input variables and constant values. A possible and frequently used fitness function is the mean of the squared errors (MSE) of the approximated values calculated by the symbolic regression model $\hat{f}(x)$ and the actually observed values y .

$$\text{MSE}(\hat{f}_x, y) = \frac{1}{n} \sum_{i=1}^n (\hat{f}(x)_i - y_i)^2 \quad (3.3)$$

The observed values y of the target function $f(x)$ include a certain amount of noise, so the error term to be minimized in symbolic regression is the sum of the error of the model ϵ_{model} and the error of the measurements ϵ_{noise} . The aim is to approximate the unknown function $f(x)$, however, the implicit measurement error ϵ_{noise} in y cannot be completely removed and is the limiting factor for the accuracy of the approximations of the model relative to the response function $f(x)$ of the studied system.

$$\begin{aligned} \hat{f}(x) &= y + \epsilon_{model} \\ y &= f(x) + \epsilon_{noise} \end{aligned} \quad (3.4)$$

Figure 3.1 shows an example for a symbolic regression model and the equivalent expression in mathematical notation.



$$\begin{aligned}
 \text{MEDV} = & -2.2045 \times 10^{-2} \text{PTRATIO} \\
 & - 1.7365 \times 10^{-6} \text{RM} \\
 & \times (-2.3469 \times 10^5 \text{RM} + 4.6022 \times 10^4 + 4.9226 \times 10^3 \text{AGE} - 3.3640 \times 10^6) \\
 & - 24.973
 \end{aligned}$$

Figure 3.1.: Example for a simple symbolic regression model and the equivalent expression in mathematical notation.

The task of symbolic regression is a very constrained and simple task for genetic programming. Symbolic regression is thus often studied in the literature as a test-bed for new operators and other extensions of GP. The scope of problems that can be solved by genetic programming is however much broader. As an example genetic programming was used to evolve classification algorithms [154]. It has to be noted, that in this approach general classification algorithms were evolved instead of classification models for a specific classification problem. In another case GP was used to automatically fix software bugs [68, 227]. In this approach genetic programming was used to evolve a bug fix in the form of a patch file for the originally incorrect source code based on a number of test cases that specified the correct behavior of the software routine. These are two examples for the power of genetic programming and are in a way more prototypical for GP than the comparatively simple task of symbolic regression. This thesis is nevertheless concerned mainly with symbolic regression and the application of symbolic regression for system identification in real-world applications.

Symbolic regression is especially useful when no or only little information about the studied system is available because GP simultaneously evolves the necessary structure and parameters of the model. This is an advantage compared to other regression methods where the model structure is often fixed and only the parameters of the model are estimated.

3.4.2. Overfitting

Overfitting can also occur in symbolic regression [220], however, the issue of generalization in GP has not yet been discussed as intensively as in the machine learning com-

munity [59], [148]. Frequently only the training quality of models generated with GP is reported, even though it has been shown repeatedly that this approach methodically invalid and might lead to overfit models that do not generalize well to new observations [83].

It can be argued that compact symbolic regression models that describe a functional relation between the input variables and the target variable are unlikely to overfit on the training. However, it has been shown that this is not true in the general case, very compact solutions can also be less robust concerning the generalization behavior as has been suggested by [56].

One approach that has been used to reduce overfitting in symbolic regression is to use an internal validation set on which all solutions of the population are evaluated. A solution that has a comparable accuracy on the training and validation set is returned as the final result [73, 232, 231]. This approach can be extended to a solution archive in which Pareto optimal (quality and size) solutions on the validation set are kept [193, 180]. The result of the GP run is the set of solutions in the archive. The advantage of this approach is that an appropriate model can be selected a-posteriori.

It is generally assumed that bloat and overfitting are related. However, it has been recently observed that overfitting can occur in absence of bloat, and vice versa. Thus, it has been suggested that overfitting and bloat are two separate phenomena in genetic programming [220], [190].

3.5. Offspring Selection

A nice description of offspring selection comparing it to similar approaches has been given by the author in [107]:

Offspring selection [2, 3] is a generic selection concept for evolutionary algorithms that aims to reduce the effect of premature convergence often observed with traditional selection operators by preservation of important alleles [4]. The main difference to the usual definition of evolutionary algorithms is that after parent selection, recombination and optional mutation, offspring selection filters the newly generated solutions. Only solutions that have a better quality than their best parent are added to the next generation of the population. In this aspect offspring selection is similar to non-destructive crossover [194], soft brood selection [10], and hill-climbing crossover [149]. Non-destructive crossover compares the quality of one child to the quality of the parent and adds the better one to the next generation, whereas offspring selection generates new children until a successful offspring is found. Soft brood selection generates n offspring and uses tournament selection to determine the individual that is added to the next generation, but in comparison to offspring selection the children do not compete against the parents. Hill-climbing crossover generates new offspring from the parents as long as better solutions can be found. The best solution found by this hill-climbing scheme is added to the next generation. The recently described hereditary

selection concept [138, 137] also uses a similar offspring selection scheme in combination with parent selection that is biased to select solutions with few common ancestors.

Offspring selection can be controlled via two parameters, namely the comparison factor and the success ratio. The comparison factor is used in the success criterion and defines a threshold for the fitness of the offspring based on the two fitness values of the parents. The comparison factor is $\in \mathbb{R}$, where a value of zero sets the threshold to the minimal fitness of all parents, and value of one sets the threshold to the maximal fitness of all parents. The success ratio is the target ratio of successful offspring in the population and is a value in the range $[0 \dots 1]$. With offspring selection new individuals are generated through crossover and mutation until the population can be filled with the correct ratio of successful and unsuccessful offspring as defined by the success ratio. A success ratio of one means all offspring must be successful. Strict offspring selection denotes offspring selection with a comparison factor of one and a success ratio of one, which means that all new offspring must have strictly higher fitness than their parents.

Genetic programming with strict offspring selection has for instance been used to produce highly accurate classifiers for medical datasets [231].

3.6. Genetic Programming with HeuristicLab

HeuristicLab is a software environment for heuristic optimization. It has been developed by members of the research group “Heuristic and Evolutionary Algorithms Laboratory” (HEAL) in Hagenberg. The chief architect of HeuristicLab is Dr. Stefan Wagner who also designed and implemented the core framework [224]. HeuristicLab is open source software and can be downloaded from <http://dev.heuristiclab.com>. The default installation of HeuristicLab also includes a genetic programming framework which has been implemented mainly by the author of this thesis and Michael Kommenda based on an earlier implementation of Dr. Stephan Winkler [3]. All experiments in this work have been prepared and executed with HeuristicLab. The GP implementation in HeuristicLab deviates in a view aspects from other GP implementations. In the following sections the specifics of the GP framework of HeuristicLab are described.

HeuristicLab is a generic environment for heuristic optimization in general and not solely for genetic programming. Two important concepts in HeuristicLab are algorithms and problems. A given problem can be configured and loaded into an algorithm, which can then be used to find a solution for the problem. The framework is designed in a way to make it possible to combine problems and algorithms freely. The default installation provides a number of different algorithms for heuristic optimization including genetic algorithm, evolution strategy, tabu search, simulated annealing, local search. GP problems like symbolic regression are implemented as problems using a symbolic expression tree encoding for Koza-style (tree-based) genetic programming. A symbolic regression problem can be solved for instance with the predefined genetic algorithm. HeuristicLab does not provide an algorithm specifically for genetic programming.

The GP framework in HeuristicLab is not strongly typed. Functions of different types must be defined explicitly in the function set, and the compatibility of functions of different types must be represented through syntactic rules. The set of all syntactic rules defines a context free grammar for symbolic expression trees. This grammar is not used to transform a linear representation to a symbolic expression tree as in grammar-based GP systems. Instead the grammar only defines the possible and valid tree shapes. Operators acting on symbolic expression trees always produce grammatically correct expressions.

The symbolic regression implementation in HeuristicLab does not support “ephemeral random constants” [101] (ERC) as described by Koza. In the original formulation of symbolic regression ERC are random constants that are added to the terminal set additionally to variable symbols for each input variable. The ERC are initialized at the beginning of the GP run and are not adapted over the GP run. GP can generate arbitrarily complex expressions to combine ERCs, so almost all possible constant values can be generated through combination of the available ERCs. In HeuristicLab constant values are directly embedded in symbolic expressions trees and are adapted over the GP run through manipulation operators similar to operators described in [183] .

4. Interpretation and Simplification of Symbolic Regression Solutions

Knowledge discovery is hindered by overly complicated models. The bloating effect of genetic programming leads to unnecessarily complex solutions which are difficult to understand and validate. Bloat is an important open topic in genetic programming [166] and a number of different theories for the cause of bloat and various methods to control bloat have been described in the literature [129, 188]. If genetic programming is used for data mining and knowledge discovery effective methods to improve parsimony and comprehensibility of GP solutions are necessary.

This topic should be addressed via multiple paths. Prevention, active reduction, and support for interactive exploration.

4.1. Bloat Control - Searching for Parsimonious Solutions

This section describes and compares algorithmic changes to genetic programming which lead to more compact and comprehensible final solutions. The different approaches are tested on a set of symbolic regression problems and compared regarding the solution quality on the test set and solution complexity. Initially two algorithms Koza-style standard GP and GP with offspring selection [2, 3] both without limits on the solution size or depth are compared. Both algorithms are then extended by the following approaches: static size and depth limits, dynamic depth limits, pruning, and dynamic operator equalization.

In the experiments in this chapter we are mainly concerned about methods to limit the size of solutions. It is possible that a deterioration in best solution quality is incurred by such methods as the search space is restricted. In this section we also always compare the best solution quality of each bloat control technique to the solution quality obtained with the reference algorithm. Best solution quality is measured on the training data only. The quality of solutions on a hold out set might differ significantly from the training quality especially if overfitting occurs. Regardless of overfitting the training quality vs. solution length ratio is an indicator for the complexity of solutions in the population and as such also relates to the complexity of a solution which is finally selected by a validation procedure. The validation procedure is necessary in any case regardless of the constraints applied to the GP process to search for compact solutions. Chapter 5 treats overfitting and countermeasures in more detail.

4.1.1. Static Depth and Length Limits

The initial generation of the population is filled with randomly created solutions which follow a certain size or depth distribution. The distribution depends on the operator that is used for initialization. In the following generations the growth of solutions can be limited by setting a static length or depth limit for the crossover and mutation operators. In the crossover operator the newly created trees are checked if they exceed the size or depth limit and if this is the case, a new offspring is created using the same parents. If it is not possible to create a new child which is smaller than the limit, one of the parents is used after a certain number of tries [101].

Static depth or length limits are a simple way to prevent unlimited growth of GP individuals. Static depth limits have been used already in [101] to limit the growth of GP trees. In the same work a rather large depth limit of 17 levels is used for most problems suggesting that it is sufficient to solve most problems.

Static limits are problematic when the average size of the population converged near the limit in later stages of the run. At that point it becomes increasingly hard for the crossover operator to produce improved offspring because the number of valid crossover points is highly restricted. A surprising result is that static size limits might even increase bloat as stated by Dignum and Poli: “[...] *size limits effectively increase the tendency to bloat since they induce more sampling of short programs, and, so, in the presence of non-flat fitness landscapes, GP populations rush towards the limit even more quickly than in the absence of the size limit!*” [53].

4.1.2. Parsimony Pressure

Parsimony Coefficient

One way to apply parsimony pressure to a population of individuals is to select parents based on an adjusted fitness value that takes the individuals size into account [101, 239]. Formally the adjusted fitness is

$$f_{adj}(i) = f(i) + c\ell(i) \quad (4.1)$$

where $f(i)$ is the raw fitness of the individual i and $\ell(i)$ is the size of the individual. The parsimony coefficient c is a constant value that determines the relative importance of the size in comparison to the raw fitness. The correct value for c is problem-dependent and it is rather difficult to find a good setting for parsimony coefficient. Intuitively it might also be beneficial to adjust c dynamically so that at a later stage of a GP run the value is different from the initial value. Thus a dynamic adjustment strategy for the parsimony coefficient has been introduced through which full control over the program length in a GP run can be exerted [163].

Covariant Parsimony Pressure

In covariant parsimony pressure [163] the parsimony coefficient is adapted based on the covariance of the individual fitness with its length. covariant parsimony pressure is

theoretically motivated and derived from the size evolution equation [162]

$$E[\mu(t+1)] = \sum_l S(G_l)p(G_l, t), \quad (4.2)$$

which describes the dynamics of expected average program size in GP. It expected mean size of programs μ at generation $t+1$ is the sum over all possible program shapes l , where $S(G_l)$ is the size of programs in the set G_l of all programs of with l . $p(G_l, t)$ is the probability of selecting programs from set G_l from the population at generation t [163].

Rewriting equation 4.2 Poli and McPhee arrive at

$$E[\Delta\mu] = \frac{\text{Cov}(\ell, f)}{\bar{f}(t)}, \quad (4.3)$$

which shows that equation 4.2 is related to Price's theorem [168]

$$\Delta Q = \frac{\text{Cov}(z, q)}{\bar{z}}, \quad (4.4)$$

which describes the dynamics of inheritable traits in biological evolution. This is quite a remarkable result as it connects GP schema theory to a theorem of biological evolution proving at the same time Price's theorem [163].

Based on this equation Poli and McPhee derive a number of different control strategies for the parsimony coefficient to control the dynamics of the average program size over a GP run. To hold the average size constant from one generation to the next the adjusted fitness function 4.5 should be used.

$$\begin{aligned} f_{adj}(x, t) &= f(x) - c(t)\ell(x, t) \\ c(t) &= \text{Cov}(\ell, f)/\text{Var}(\ell) \end{aligned} \quad (4.5)$$

It has been shown that the average length of individuals in the population can be tightly controlled through covariant parsimony pressure and that the target average program length can be controlled to follow a given function over the GP run (e.g. sine).

Poli and McPhee have suggested to turn covariant parsimony pressure on and off at specific generations of a GP run and to let the population grow freely in between phases where parsimony pressure is applied [163]. This suggestion is taken up in this work in Chapter 5 where we introduce a novel overfitting prevention mechanism which activates covariant parsimony pressure only when overfitting is detected in a GP run. The idea is that in an overfitting phase the average length of programs in the generation should not grow. At such stages covariant parsimony pressure can be used, to keep the average code length constant or gradually reduce the average program size. As long as no overfitting occurs, parsimony pressure is deactivated and the population is allowed to grow freely.

With parsimony pressure there is a risk that selection probability becomes mainly driven through the parsimony pressure and the fitness of the individual is less relevant. If this happens it is expected that the final solution quality produced by such runs is worse than the solution quality that would be achieved with hard limits. In [163] it has been

shown for simple regression problems (single-variate polynomial of 6th degree and 9th degree) that solution quality is not negatively effected by covariant parsimony pressure. Such polynomial functions are often used as benchmark problems for symbolic regression but they are only very simple test cases. It has not yet been analyzed how covariate parsimony pressure works for real world problems. In this work we also apply covariant parsimony pressure on real word problems; the experimental results are presented later in this chapter.

The theory behind covariant parsimony pressure is based on fitness proportional selection. In [163] it has been demonstrated that for benchmark problems the method also works when tournament selection with group size of two is used. We observed that with tournament selection the trajectory of average program sizes are not as stable as with proportional selection. In particular, because the average program size can only be controlled to assume a given value in expectations there are fluctuations in the actual average program size. The fluctuations can have the effect that the whole population is forced down to the minimal program size in only a few generations in extreme cases. We observed such effects especially when combining covariant parsimony pressure with tournament selection, however, we did not pursue this yet, further experiments are necessary. In the experiments presented later in this section we used fitness proportional selection only in combination with covariant parsimony pressure.

Lexicographic Parsimony Pressure

In [128] a variant of tournament selection has been described which select individuals not only based on their fitness but also on their size. Lexicographic tournament selection works in the same way as tournament selection, only that it prefers individuals with smaller size if they have the same quality. It has been suggested in [129] that a large number of different fitness values are possible in symbolic regression because changes in the lowest parts of the trees cause minor changes in fitness. Two bucketing variants have been proposed for such situations [128]. Individuals are assigned to buckets based on their fitness and individuals from the same bucket are treated in lexicographic tournament selection as if they have the same fitness value.

In this work lexicographic tournament selection is not analyzed in greater detail. In most experiments presented in the following sections, standard tournament selection is used and bloat is controlled by other mechanisms.

4.1.3. Dynamic Depth Limits

Recent work describes how depth-limits can be varied dynamically based on fitness [188]. Initially the dynamic depth limit (DDL) is set to small value. When a new best solution is found which exceeds the dynamic depth limit it is set to the depth of the new best solution. If the best solution is smaller than the dynamic depth limit it can be decreased again. In this case some individuals in the current population might exceed the depth limit. Through the application of crossover, however, the individuals of the next generation are limited to the new reduced depth limit. If no reproduction is used

the depth of all individuals of the next generation is within the depth limit, otherwise a few generational steps are necessary until all individuals of the population are within the bounds again.

The advantage of dynamic depth limits in comparison to static limits is that the limit is adapted by the algorithm as necessary for the problem. In the case of static limits solutions exceeding the limit are rejected in all cases even when they improve the best known solution. With dynamic depth limits the limit is adapted accordingly when a new best of run solution is found. Even though program growth is not prohibited because the dynamic limit can potentially be extended, the approach still effectively prevents program growth which is not matched with a proportional improvement in solution quality.

Dynamic Depth Limits for Offspring Selection

Dynamic depth limits have been combined with offspring selection [238]. For each newly created offspring first the offspring selection success criterion is checked. If the offspring is successful, the dynamic depth limit is adapted based on the quality and the depth of the offspring. The dynamic depth limit is decreased if the offspring has a smaller depth and is better than the best solution (on the training) so far. All offspring not exceeding the depth limit are accepted. If offspring exceeds the dynamic depth limit then it is only accepted if it has better quality than the best solution so far. In this case the dynamic depth limit is set to the depth of the new offspring and the offspring is accepted. If the offspring exceeds the depth limit and it is worse than the best solution so far then the offspring is rejected. In order to prevent fluctuations in the dynamic depth limit any increase or decrease of the dynamic depth limit must be backed by a significant improvement in solution quality. This can be controlled via parameters c_{lower} and c_{Raise} . Zavoianu suggests values of 3% and 1.5%, respectively [238].

Algorithm 1 describes how the dynamic depth limit is adapted based on the depth and quality of a newly created offspring. This algorithm is derived from the original formulation of DDL [186, 188].

In [238] OSGP with dynamic depth limits has been applied to two real world problems and it has been shown that the dynamic depth limit leads to smaller final solutions with comparable model quality. In this work we compare the final solutions produced by SGP and OSGP, both extended with dynamic depth limits on benchmark problems and additional real world problems.

In the original formulation of Silva and Costa [188] an offspring is replaced by one of its parents, if it is rejected because of the dynamic depth limit. In the adaption for offspring selection [238] this is not the case. Instead new parents are selected and crossover is applied again. The strategy potentially amplifies the effect of crossover bias. It has been suggested that it is better to accept the parents if crossover produces illegal offspring [188].

Algorithm 1: Dynamic depth limits for offspring selection

```
if SuccessfulOffspring then
  if Maximization then
    | relativeQuality  $\leftarrow$  (quality / bestQuality) - 1 ;
  else
    | relativeQuality  $\leftarrow$  (bestQuality / quality) - 1;
  end
  if  $tree_{depth} \leq ddl$  then
    | // depth is smaller than dynamic limit => reduce limit if the
    |   quality improvement is large enough
    | if relativeQuality  $\geq c_{Lower}(ddl - tree_{depth})$  then
    |   | ddl  $\leftarrow$  Max( $tree_{depth}$ , InitialDepthLimit);
    |   end
  else
    | // depth is larger than dynamic limit => increase limit if
    |   the quality improvement is large enough
    | if relativeQuality  $\geq c_{Raise}(tree_{depth} - ddl)$  then
    |   | ddl  $\leftarrow tree_{depth}$ ;
    |   else
    |     | // depth is larger but no improvement => reject
    |     | SuccessfulOffspring  $\leftarrow$  false;
    |   end
  end
end
end
```

Parameter	Value
Population size	500
Generations	50
Selection	Offspring selection 50% Proportional 50% Random Comparison factor = 1.0 Success ratio = 1.0
Replacement	Generational 1-Elitism
Initial length	3 ... 100
Initial depth limit	7
Max depth limit	25
Static size limit	1000
Crossover	Sub-tree swapping
Mutation	Single-point Replace branch
Mutation rate	15%
Function set	$+, -, \frac{x}{y}, \times$
Fitness function	R^2

Table 4.1.: Parameter settings for the OSGP-DDL experiments.

4.1.4. Experiments

We applied OSGP with DDL (OSGP-DDL) on three different datasets. Friedman-I (see A.3.1) and Breiman-I (see A.3.1) are artificial datasets. The Breiman-I function includes a conditional and is harder to approximate with symbolic regression than the Friedman-I function. The Chemical-I dataset (see A.3.2) is a dataset from a real world industrial chemical process. Table 4.1 shows the GP parameter setting for the experiments. The results are compared to the results of OSGP with static limits on the same datasets (see 4.1.7).

Results

In Figure 4.1 the change in best solution quality and the amount of bloat shown for three datasets Friedman-I, Breiman-I, and Chemical-I. The amount of bloat relates the average quality $\bar{f}(g)$ at generation g with the average tree length $\bar{l}(g)$ at generation g

and is calculated using equation 4.6 [220] (also see Chapter 3).

$$\begin{aligned}
\text{bloat}(g) &= \frac{\Delta \bar{l}(g)}{\Delta \bar{f}(g)} \\
\Delta \bar{f}(g) &= \frac{\bar{f}(g) - \bar{f}(0)}{1 + \bar{f}(0)} \\
\Delta \bar{l}(g) &= \frac{\bar{l}(g) - \bar{l}(0)}{\bar{l}(0)}
\end{aligned} \tag{4.6}$$

In all experiments in this chapter we are mainly interested in the fitness on the training set. All qualities that are reported in the results are the squared correlation coefficient (R^2) on the training set. In Chapter 5 more experiment results are presented that focus on the generalization ability of different GP configurations.

Figure 4.1 shows that because of dynamic depth limits the amount of bloat is greatly reduced in comparison to OSGP with static limits. The best solution quality is not negatively effected by the application of dynamic depth limits.

The best solution quality and amount of bloat for the last generation are also given in Tables 4.4, 4.5, and 4.6 later in this chapter.

4.1.5. Operator Equalization

Operator equalization is an bloat control measure motivated by the crossover bias theory of bloat [54]. To counteract the bias of crossover to produce more smaller offspring an equalization step is introduced after crossover. Offspring created from crossover are filtered before they are accepted into the next generation of the population. Algorithm 2 shows the general method of filtering offspring based on an acceptance criterion. Offspring selection, dynamic depth limits, and operator equalization all plug into the GP process in the same way repeatedly creating new offspring until the population is full.

Algorithm 2: Filtering of offspring in the formulation of dynamic depth limits and operator equalization

```

while Population is not full do
    Select parents;
    Create offspring;
    if Accept(offspring) then
        | Add offspring to next generation of population;
    end
end

```

The difference lies in the way how the acceptance criterion is defined. In the case of operator equalization the filter effectively controls the length distribution of individuals in the population by limiting the number of individuals of a given size in the population. The basis of operator equalization is a target length distribution histogram. Individuals

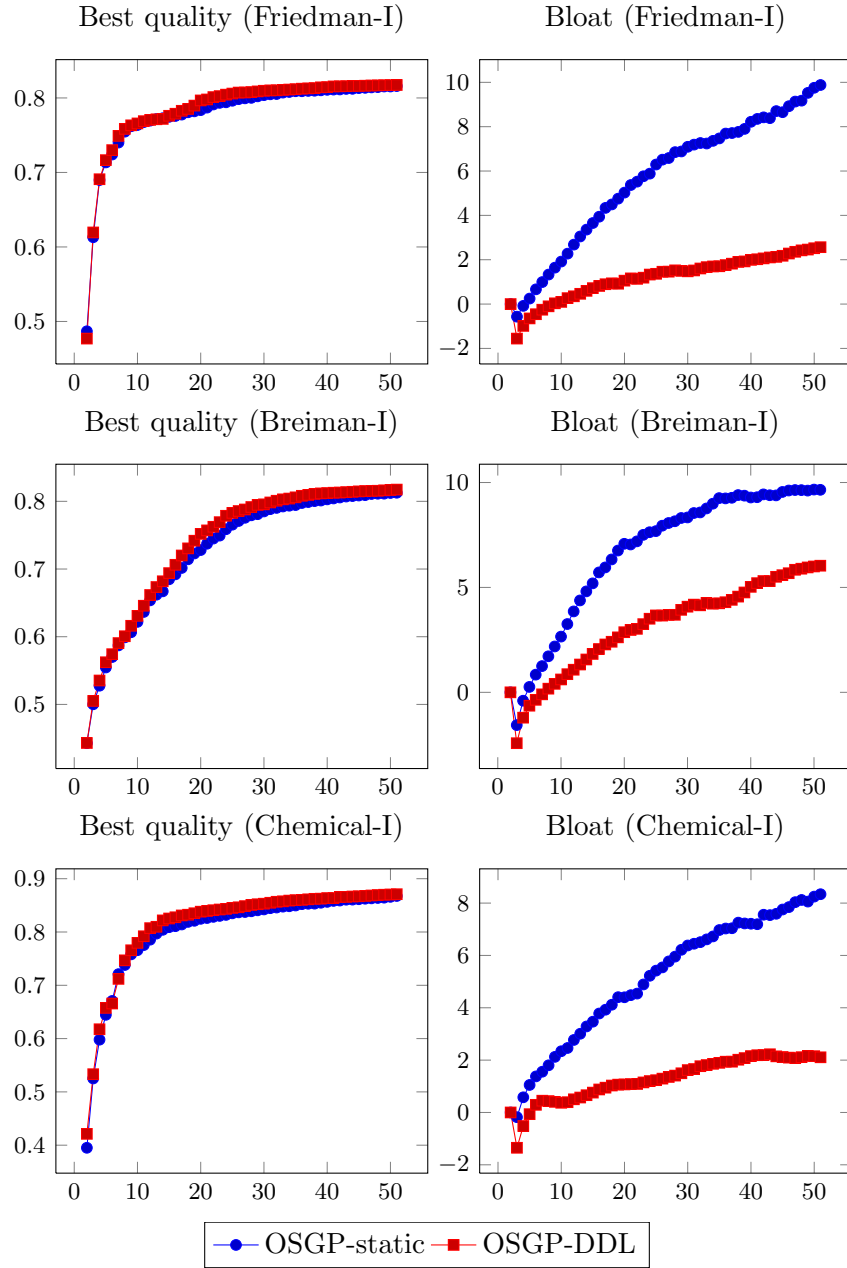


Figure 4.1.: Best solution quality and bloat comparison of OSGP with static limits and OSGP with dynamic depth limits (average values over thirty independent runs, x-axis is generation index).

are grouped by length into several bins using equation 4.7. The number of individuals that can be accepted into each bin is limited by a static bin capacity. The target size distribution of solutions in the population can be configured freely through manipulation of the bin capacities.

$$b \leftarrow \lfloor \frac{\text{treeLength} - 1}{\text{binSize}} \rfloor + 1 \quad (4.7)$$

It has been shown empirically that genetic programming with operator equalization is bloat free [54]. This is strong empirical evidence for the crossover bias theory of bloat. It should be noted that individuals are evaluated only after they have been accepted into the population by the equalization filter, otherwise a lot of effort is wasted evaluating individuals which are discarded by the equalization filter after fitness evaluation.

Dynamic Operator Equalization

Operator equalization is a method to control the length distribution of individuals in the population exactly by filtering offspring based on their length. The weakness of operator equalization is that the target distribution and the maximal size limit which is appropriate for a given problem is not known a-priori. Thus the method has been improved to adapt the target distribution and the maximal size limit dynamically based on the fitness and length of individuals in the population [188, 189, 190, 191].

Dynamic operator equalization combines dynamic depth limits and operator equalization. Instead of the static target distribution the initial target distribution is created with bin capacities to hold all solutions of the initial generation. In the following generations the bin capacity is adapted based on the average fitness of solutions in each bin in relation to the average fitness over the whole population using equation 4.8. The capacity of bins, that hold individuals that are fitter on average than the whole population, is increased while the capacity of bins, that hold individuals that are on average worse than the population, is accordingly decreased.

$$\text{binCapacity}_i = \text{round}(\text{PopSize} \frac{\bar{f}_i}{\sum_j \bar{f}_j}) \quad (4.8)$$

The algorithm for acceptance of an offspring is shown in Algorithm 3. For each new offspring the target bin is determined based on its length. The offspring is accepted into the population only when the bin is not yet full. The only exception is if the new offspring has a better fitness than the best solution in that bin so far. In this case the offspring is accepted even though the bin is already full and the capacity of the bin is increased.

The acceptance criterion shown in Algorithm 3 combined with the adaption of bin capacities leads to dynamic adaption of the target distribution based on solution fitness. Similarly to dynamic depth limits dynamic operator equalization an extension of the size distribution is possible when a new best of run solution is found for which a bin does not yet exist. In this case a new bin of capacity one is created and the solution is added to the bin. If necessary bins in between the newly created bin and the last existing bin are created with a capacity of one.

Algorithm 3: Acceptance criterion of dynamic operator equalization

```
input : b: bin index, tree: current offspring  
accept  $\leftarrow$  false;  
if Exists(b) then  
    // increase bin count of bin b when tree is accepted  
    if IsNotFull(b) or NewBestOfBin(tree) then  
        AddToBin(b, tree);  
        accept  $\leftarrow$  true;  
    end  
else  
    // add new bins when a larger best-of-run tree is found  
    if NewBestOfRun(tree) then  
        CreateBin(b);  
        AddToBin(b, tree);  
        accept  $\leftarrow$  true;  
    end  
end
```

It has been observed that for some problems a large number of surplus evaluations are necessary before the population can be filled [189]. In contrast to the static operator equalization in the original formulation of dynamic operator equalization all individuals are also evaluated in order to accept best-of-bin individuals. This has the effect that many solution candidates are evaluated first before they are rejected. To reduce the number of fitness evaluations of individuals that are rejected anyway Silva and Dignum suggest to change the process to discard individuals for which a bin exists but is already full in all cases. Individuals for which a bin does not yet exist are evaluated and a new bin is created if a new best of run solution is found. This has the drawback that some solutions are rejected without evaluation and this could lead to rejection of best-of-run individuals. The benefit, however, is a large performance gain.

Recently a different approach to improve performance in dynamic operator equalization has been proposed [190]. Here the number of evaluation is reduced through dynamically mutating individuals until they fit an empty bin. The results suggest that the mutation approach is better when overfitting occurs [190]. In this work we do not treat mutation operator equalization in depth.

Dynamic Operator Equalization and Offspring Selection

The formulation of dynamic operator equalization resembles offspring selection in that newly created offspring are filtered. The difference is that the filter which is defined over the parents of the offspring in the case of offspring selection is instead defined over the target distribution and its current fill level in the case of dynamic operator equalization. It is tempting to try to combine offspring selection and dynamic operator equalization.

Algorithm 4 describes the combined method of offspring selection and dynamic operator equalization. The difference to the original formulation of dynamic operator equalization is that individuals are accepted only if they meet the offspring selection success criterion. For each offspring first the bin of the offspring is checked. If a bin already exists and it is not full then the individual is evaluated and accepted if it meets the OS success criterion. If the bin is already full the offspring is discarded without evaluation. If the bin does not yet exist the individual is evaluated and accepted if it is the new best of run individual. The target distribution is extended if necessary.

Algorithm 4: Acceptance criterion of dynamic operator equalization with offspring selection

```

input : b: bin index, tree: current offspring
accept  $\leftarrow$  false;
if Exists(b) then
    if IsNotFull(b) and MeetsSuccessCriterion(tree) then
        AddToBin(b, tree);
        accept  $\leftarrow$  true;
    end
else
    if NewBestOfRun(tree) and SignificantImprovement(tree) then
        CreateBin(b);
        AddToBin(b, tree);
        accept  $\leftarrow$  true;
    end
end

```

Experiments

We applied SGP with dynamic operator equalization (SGP-DynOpEq) on three datasets to analyze the ability of the method to control bloat. The algorithm has been applied to the same three datasets as in the previous experiments (Breiman-I, Friedman-I, and Chemical-I). Additionally we also executed GP runs with covariant parsimony pressure (SGP-CPP) on the same datasets for comparison. The results of both algorithms are compared to SGP with static limits (SGP-static) and SGP with dynamic depth limits (SGP-DDL). Table 4.2 lists the parameter settings for SGP-DynOpEq, SGP-DDL, and SGP-CPP. The parameter settings for SGP-static are given in Table 4.3 the static length limit is 250 nodes and the static depth limit is 17 levels.

Results

Figure 4.2 shows a comparison of the best solution quality and bloat behavior for SGP with static size constraints (SGP-static), SGP with dynamic operator equalization (SGP-DynOpEq), SGP with dynamic depth limits (SGP-DDL), and SGP with covariant par-

Parameter	SGP-DynOpEq	SGP-DDL	SGP-CPP
Population size	2000	2000	2000
Generations	50	50	50
Selection	Tournament	Tournament	Tournament
Group Size	7	7	6
Replacement	Generational	Generational	Generational
	1-Elitism	1-Elitism	1-Elitism
Initial length	3 ... 100	3 ... 100	3 ... 150
Initial depth	1...7	1...7	1...8
Max depth limit	25	25	25
Static size limit	1000	1000	1000
Crossover	Sub-tree swapping	Sub-tree swapping	Sub-tree swapping
Mutation	Single-point	Single-point	Single-point
	Replace branch	Replace branch	Replace branch
Mutation rate	15%	15%	15%
Function set	$+, -, \frac{x}{y}, \times$	$+, -, \frac{x}{y}, \times$	$+, -, \frac{x}{y}, \times$
Fitness function	R^2	R^2	R^2

Table 4.2.: Parameter settings for SGP-DynOpEq, SGP-DDL, and SGP-CPP experiments.

simony pressure (SGP-CPP) for three problems. The results shown are average values over thirty independent GP runs for each configuration.

The bloat limiting effect of dynamic operator equalization can be clearly seen in the charts. Both SGP-DynOpEq and SGP-DDL result in a significant reduction in bloat compared to SGP-static. Notably in the first generations the amount of bloat is higher with DynOpEq, because the average program size is quickly increased to a certain level, while the average quality does not improve so quickly. With SGP-DDL there is a continuous but slow increase bloat even at the later stages of the GP run. In comparison the amount of bloat does no increase in the later stages of the GP runs with DynOpEq.

The effect that the average solution size decreases in the early stages of the run while the average quality improves, that can be often observed for symbolic regression problems, is completely removed by operator equalization. In terms of solution quality the application of dynamic operator equalization causes a slight deterioration in solution quality compared to SGP-static. The same deterioration can also be observed for SGP-DDL. Covariant parsimony pressure does not bloat at all but also produces significantly worse solutions. This is likely caused by the combination of tournament selection with covariant parsimony pressure, this effect should be analyzed in further experiments.

The results are also summarized in Tables 4.4, 4.5, and 4.6.

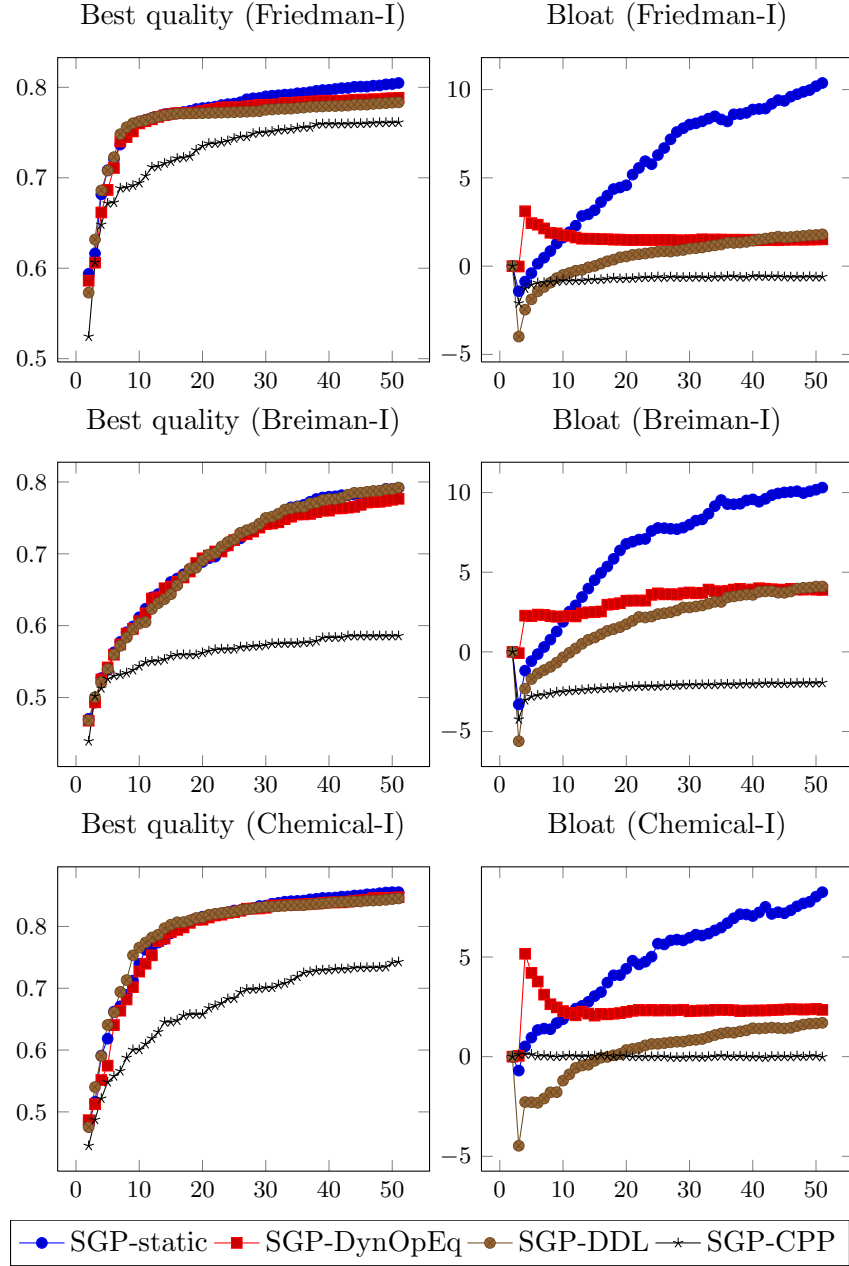


Figure 4.2.: Best solution quality and bloat comparison of SGP-static, SGP-DynOpEq, SGP-DDL, and SGP-CPP (average values over thirty runs, x-axis shows generation index).

4.1.6. Pruning to Reduce Bloat

Pruning in tree-based genetic programming means to remove branches from GP solutions that have no or only minimal effect on the quality of the solution. Pruning is also frequently used in other learning methods with tree-based model representation for instance CART [28] and MARS [71], where it is often used to reduce overfitting. Pruning can be used to reduce bloat by removing inviable code from individuals over the whole population [8, 3, 97]. To make pruning viable it must be possible to determine the effect of removing a branch efficiently. For some GP problems it is possible to determine the effect of branch-removal statically without fully evaluating the individual. For instance for the artificial ant problem turning left and in the next operation turning right is unoptimized code which can be removed because through semantic analysis this is obviously useless. Similar patterns can be defined for symbolic regression (e.g. $x * 0$). However, the impact of static pruning is limited since the number of such patterns is very large and it is practically impossible to define all patterns which might lead to inviable or unoptimized code. In the case of symbolic regression the relevance of branches can be determined by comparing the output of the original tree with a pruned tree. If the output of the pruned tree is almost the same then the pruned tree still contains the important code.

When pruning is included into the GP process a number of aspects must be considered. Since pruning is applied frequently it should be rather efficient so that the process does not become too slow. Also pruning should not remove too much genetic diversity. We observed that a certain amount of unoptimized code in the population is necessary for the evolutionary process. If pruning is applied too vigorously there is a risk of premature convergence because of the sudden reduction of genetic diversity.

In practical applications we found a greedy pruning strategy based on branch impact metrics (see Section 6.2.6) shown in Algorithm 5 to work sufficiently well. The greedy method evaluates all branch impacts and then removes the branch with smallest impact preferring larger branches. This is done iteratively until a maximal quality deterioration is reached or the size of the tree has been reduced by a certain factor. A similar approach for numerical simplification based on a number of additional metrics for the relevance of tree-branches is proposed in [97, 88].

The disadvantage of the greedy pruning strategy is that certain types of unoptimized code cannot be detected or removed. Since the method removes one branch at a time it is not possible to detect cases of interaction between two separate branches, where removing one of them causes a big change in solution quality, even though removing both simultaneously would not change the quality at all. Such cases are however rather difficult to detect and can practically only be removed through exhaustive methods which cannot be included in the GP process because of performance considerations.

The calculation of the branch impact is rather expensive as the branch has to be evaluated to calculate a replacement value and then the manipulated tree must be evaluated to determine the effect on the quality. To make the method more efficient, only a limited number of branches is considered for removal in every iteration (tournament size) and the branch as well as the manipulated tree are not evaluated on the full training set,

Algorithm 5: Greedy iterated tournament pruning

Data: t : original tree, iterations, groupSize
Result: prunedTree
prunedTree \leftarrow tree;
while $c \leq \text{iterations}$ **do**
 $B \leftarrow \text{SelectRandomBranches}(\text{prunedTree}, \text{groupSize})$;
 prunedBranch $\leftarrow \operatorname{argmin}_{b \in B} \text{Impact}(b) / \text{RelativeSize}(t, b)$;
 Inc(c);
end

$$\text{RelativeSize}(t, b) = \text{Size}(t) / (\text{Size}(t) - \text{Size}(b))$$

but instead only on a limited number of random samples from the training set. This is especially important if the training set contains a large number of samples.

4.1.7. Does Offspring Selection Reduce Bloat?

Offspring selection with strict parameter settings enforces that newly created offspring must improve upon its parents quality. Because of this, offspring selection should intuitively have a limiting effect on bloat because it has a negative bias against introns (inviatile code). In the light of crossover bias theory [164, 52], however, there is no indication that offspring selection should reduce bloat. Instead the effect of crossover bias could be amplified, because parent selection and crossover are executed repeatedly until the population can be filled with successful offspring. Sub-tree crossover produces more smaller offspring then larger offspring. Smaller offspring tendentially have lower fitness. This leads to a bias to accept more larger offspring into the population. If sub-tree crossover is repeatedly applied until a successful individual is found the effect of crossover bias is amplified. In this work the hypothesis that OS reduces bloat is tested for the first time and the effect of offspring selection on bloat is analyzed.

In symbolic regression inviatile code is unlikely to occur if function sets with only arithmetic operators are used [186, 187]. The bloating effect is caused mainly by the growth of unoptimized code. If conditional and boolean functions are included in the function set, inviatile code can be produced by the process much more easily. In light of these observations, we also compare the amount of bloat of both SGP and OSGP with a small function set including only arithmetic operators and with an extended function set also including conditionals and boolean operators. If the hypothesis holds that offspring selection reduces bloat because of its bias against inviatile code, then the difference in bloat between OSGP and SGP should be even larger with the extended function set. The amount of bloat observed with the extended function set should be smaller than the amount of bloat observed with the arithmetic function set.

Parameter	SGP	OSGP
Population size	2000	500
Generations	50	50
Selection	Tournament Selection Group size = 7	Offspring selection 50% Proportional 50% Random Comparison factor = 1.0 Success ratio = 1.0
Replacement	Generational 1-Elitism	Generational 1-Elitism
Initial length	3 ... 100	3 ... 100
Max. initial depth	12	12
Crossover	Sub-tree swapping	Sub-tree swapping
Mutation rate	15%	15%
Mutation	Single-point Replace branch	Single-point Replace branch
Fitness function	R^2	R^2

Table 4.3.: Genetic programming parameter settings for the analysis of bloat in SGP and OSGP.

Experiment Setup

Two types of experiments are executed. First standard GP (SGP) and GP with offspring selection (OSGP) are applied to a number of test problems without constraints on the program size. Then the same experiments are repeated with static size limits for both algorithms.

In the experiments without size constraints four different configurations are tested: SGP with arithmetic functions only (SGP), SGP with an extended function set including conditionals and boolean operators (SGP-full), OSGP with arithmetic functions (OSGP), and OSGP with the extended function set (OSGP-full).

Additionally the four same experiments are also executed using static depth and length limits (SGP-static, SGP-static-full, OSGP-static, and OSGP-static-full). The parameter settings of both algorithms are given in Table 4.3. For the experiments without size constraints the depth and length limits are set to very large values (100, 100000) to make sure that the limit is not reached over the whole run. For the experiments with static constraints the depth limit was set to 17 levels and the length limit was set to 250 nodes.

For the experiments we use three different datasets. Friedman-I (see A.3.1) and Breiman-I (see A.3.1) are artificial datasets. The Breiman-I function includes a conditional and is harder to approximate with symbolic regression than the Friedman-I function. The Chemical-I dataset (see A.3.2) is a dataset from a real world industrial chemical process.

Results

The results presented below show the trajectories of the best solution quality and the amount of bloat (see 3) over the whole run averaged over thirty independent runs.

Results Without Size Limits

Figure 4.3 shows the best quality and amount of bloat for the Friedman-I, Breiman-I, and Chemical-I datasets without limits on the program length or depth.

OSGP produces better solutions than SGP for all problems especially with the extended function set (OSGP-full). Surprisingly OSGP-full also bloats fastest with the Friedman-I and Chemical-I datasets. With the Breiman-I dataset OSGP using only arithmetic functions bloats fastest. In general OSGP bloats fastest with all datasets. Even though it produces better solutions, the larger growth in solution length is not matched by a proportional growth in solution quality.

Another interesting observation is that OSGP bloats faster with the full function set than with the arithmetic function set. With the full function set inviable code is much more likely than with the arithmetic function set. Even though strict OSGP as it is used in this experiment has a bias against inviable code the bloating effect is stronger. This contradicts the hypothesis that OSGP reduces bloat because of a bias against inviable code.

The Breiman-I function includes a conditional and is more difficult to approximate with GP. This can also be observed in the results of the experiments.

The results for the final generation are also given in Tables 4.4, 4.5, and 4.6 that show a comparison of all experiment results in this chapter for all datasets.

Results With Static Size Limits

Figure 4.4 shows the best quality and amount of bloat for the Friedman-I, Breiman-I and Chemical-I problems with static limits on the program length and depth. The static limits for length and depth are 250 nodes and 17 levels, respectively.

The results are similar to the results of the experiments without size limits. OSGP produces solutions with a higher fitness than SGP especially when using the full function set. OSGP-full also bloats faster than the other configurations with all datasets. Compared to the results without size constraints the amount of bloat is less for all configurations because of the size constraints but the relative differences between algorithms are similar.

For the Breiman-I problem the configurations with the full function set bloat faster than the configurations with only arithmetic functions. Notably, SGP-full bloats produces more bloat than OSGP with the small function set for this dataset.

The results for the final generation are also given in Tables 4.4, 4.5, and 4.6.

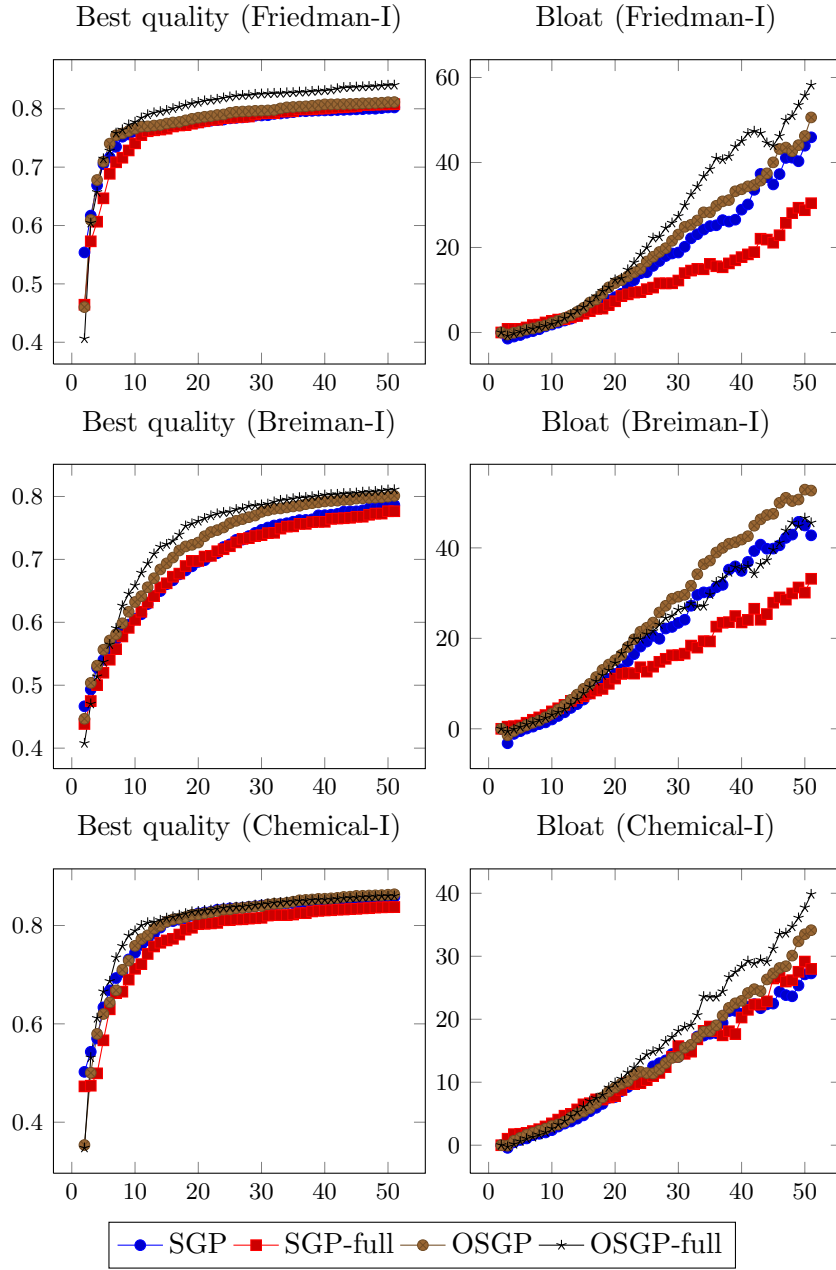


Figure 4.3.: Best quality and bloat of SGP and OSGP without size limits (average values over thirty independent GP runs, x-axis shows number of generations).

Conclusion

Interpreting the results of the experiments shown in Figures 4.3 and 4.4 the intuitive assumption that offspring selection reduces bloat does not hold. Instead we observed that average program length grows faster when offspring selection is used, both with and without size constraints. The average quality of solutions is higher when offspring selection is applied but this is combined with a disproportional increase in solution size. This has also been suggested already in [238]. The results of these experiments provide additional evidence for the crossover bias theory of bloat.

4.1.8. Multi-objective GP

Multi-objective optimization approaches have also been used for bloat control [22, 45, 44, 153]. Instead of including the solution complexity as a penalization term into the fitness value the multi-objective approach uses a multi-dimensional fitness value which includes a component for accuracy and a component for complexity. The aim of the algorithm is to produce a Pareto-optimal front of solutions for both fitness components.

One of the most well known multi objective optimization algorithm is the non-dominated sorting genetic algorithm (NSGA-II) [47, 48]. NSGA-II has for instance been used for to search for parsimonious symbolic regression models in [94].

In this work we do not treat the topic of multi-objective optimization in full detail and a single-objective GP algorithm is used for all experiments.

4.1.9. Summary of Results of Bloat Experiments

Tables 4.4, 4.5, and 4.6 summarize the results of all experiments with different bloat control methods in this chapter. The results for SGP and OSGP without any constraints on program size are also given for comparison. Size is the average program length in the last generation. Best solution quality is the squared correlation coefficient (R^2) in the last generation on the training set. The bloat value is the value of the last generation calculated using equation 4.6. All values are averages over 30 independent GP runs the confidence interval is given for $\alpha = 0.05$ and assumes normally distributed values.

4.1.10. Effects of Bloat Control on Genetic Diversity

Bloat control methods can have a strong effect on the dynamic of the GP process and might interfere with genetic diversity. If the bloat control method causes a sudden reduction of genetic diversity from one population to the next it is possible that this ultimately leads to premature convergence. Issues regarding bloat control and genetic diversity have been discussed in [239, 45, 44] and more recently also in [9]. If bloat control measures are used it is recommended to also observe the genetic diversity in the population and if necessary counteract loss (e.g. by increasing the mutation rate).

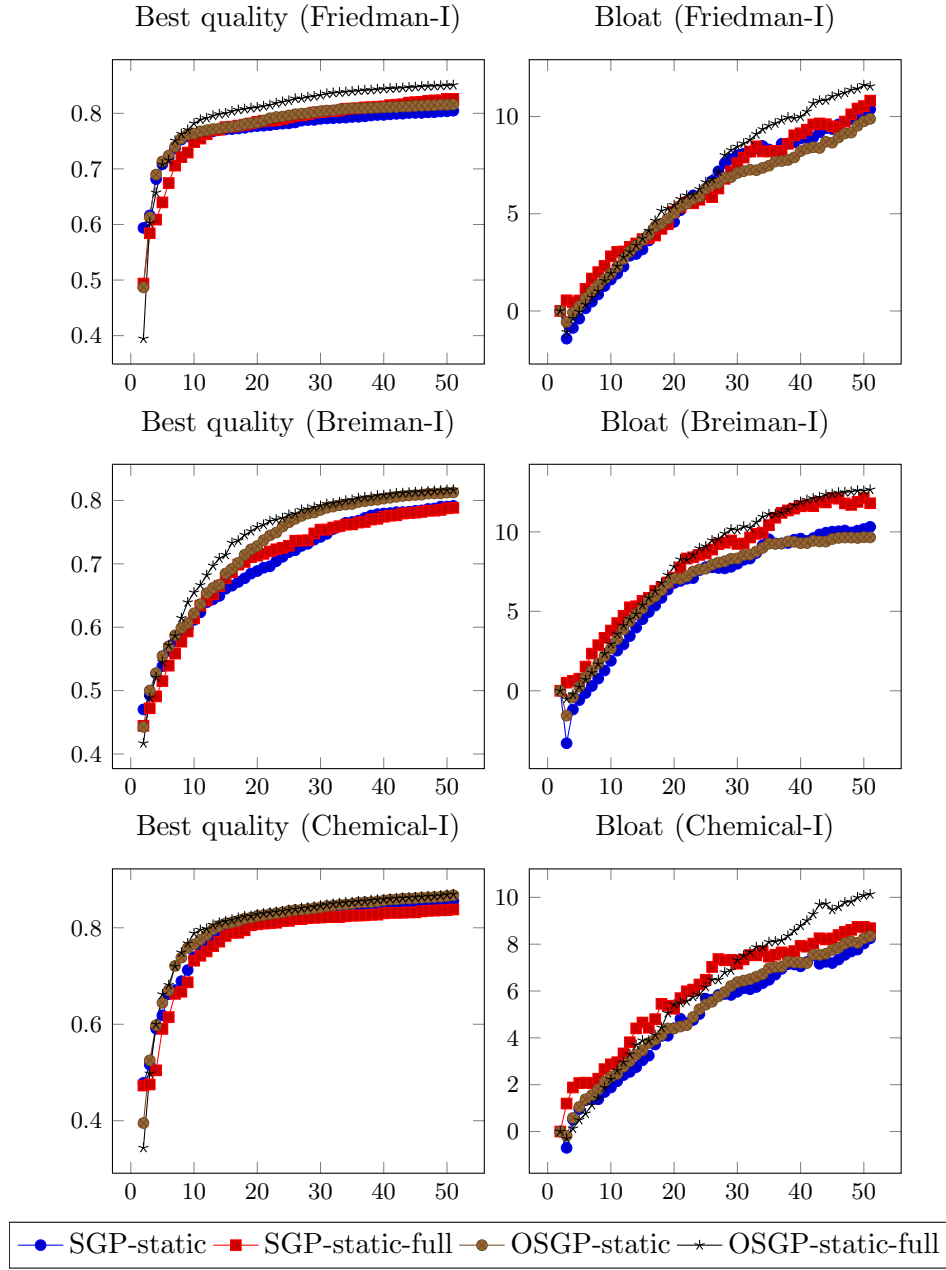


Figure 4.4.: Best quality and bloat of SGP and OSGP with static size limits (average values over thirty independent GP runs, x-axis shows number of generations).

Algorithm	Size		R^2 (train)		Bloat	
SGP	888.5	± 131.7	0.78	± 0.01	47.93	± 7.17
SGP-full	572.7	± 112.2	0.77	± 0.01	36.52	± 7.35
OSGP	1,145.0	± 163.2	0.80	± 0.00	55.31	± 8.09
OSGP-full	997.2	± 231.2	0.81	± 0.00	59.21	± 14.15
SGP-static	196.6	± 10.5	0.79	± 0.01	10.26	± 0.64
SGP-static-full	183.1	± 14.1	0.78	± 0.01	11.49	± 0.97
OSGP-static	222.0	± 7.5	0.81	± 0.00	9.45	± 0.34
OSGP-static-full	226.7	± 10.3	0.82	± 0.00	12.23	± 0.63
OSGP-DDL	160.0	± 20.1	0.82	± 0.00	6.49	± 1.00
SGP-DynOpEq	79.8	± 5.7	0.78	± 0.01	4.18	± 0.51
SGP-CPP	32.5	± 1.6	0.59	± 0.01	-1.95	± 0.05
SGP-DDL	96.2	± 10.4	0.78	± 0.01	4.32	± 0.59

Table 4.4.: Summary of results of bloat experiments for the Breiman-I problem (averages over 30 runs, confidence intervals for $\alpha = 0.05$).

Algorithm	Size		R^2 (train)		Bloat	
SGP	940.1	± 120.8	0.80	± 0.00	48.17	± 6.08
SGP-full	581.1	± 157.5	0.81	± 0.00	35.09	± 9.37
OSGP	1,115.1	± 149.1	0.81	± 0.00	54.72	± 7.68
OSGP-full	1,139.7	± 168.4	0.84	± 0.00	64.58	± 9.60
SGP-static	192.6	± 10.4	0.80	± 0.00	9.77	± 0.65
SGP-static-full	180.9	± 11.5	0.82	± 0.01	10.36	± 0.73
OSGP-static	219.6	± 8.7	0.82	± 0.00	9.56	± 0.43
OSGP-static-full	216.6	± 9.3	0.85	± 0.00	11.11	± 0.53
OSGP-DDL	79.0	± 9.5	0.82	± 0.00	2.56	± 0.48
SGP-DynOpEq	50.6	± 1.5	0.79	± 0.00	1.64	± 0.13
SGP-CPP	70.4	± 3.0	0.76	± 0.00	-0.60	± 0.09
SGP-DDL	60.6	± 8.9	0.79	± 0.00	1.97	± 0.52

Table 4.5.: Summary of results of bloat experiments for the Friedman-I problem (averages over 30 runs, confidence intervals for $\alpha = 0.05$).

Algorithm	Size		R^2 (train)		Bloat	
SGP	655.1	± 108.6	0.86	± 0.00	31.13	± 5.15
SGP-full	481.4	± 83.1	0.84	± 0.01	29.38	± 4.96
OSGP	786.7	± 108.4	0.86	± 0.00	34.55	± 4.75
OSGP-full	786.4	± 125.0	0.86	± 0.00	42.22	± 6.61
SGP-static	166.2	± 14.2	0.85	± 0.00	8.05	± 0.83
SGP-static-full	146.4	± 12.4	0.84	± 0.00	8.71	± 0.83
OSGP-static	203.1	± 11.3	0.87	± 0.00	7.95	± 0.52
OSGP-static-full	195.2	± 13.9	0.87	± 0.00	9.53	± 0.76
OSGP-DDL	84.2	± 10.5	0.87	± 0.00	2.51	± 0.45
SGP-DynOpEq	62.5	± 3.6	0.84	± 0.00	2.40	± 0.26
SGP-CPP	92.2	± 4.7	0.74	± 0.02	0.03	± 0.16
SGP-DDL	54.4	± 7.4	0.84	± 0.00	1.73	± 0.44

Table 4.6.: Summary of results of bloat experiments for the Chemical-I problem (averages over 30 runs, confidence intervals for $\alpha = 0.05$).

4.2. Simplification of Symbolic Regression Solutions

Guiding genetic programming to search for compact solutions is one way to improve the comprehensibility of final solutions. In spite of such bloat control methods the final solution produced by the algorithm is still likely to contain inviable or unoptimized code. Thus methods to analyze and simplify solutions a-posteriori are needed. Visual support for the analysis of GP solutions can improve the interpretation of solutions in practical applications.

4.2.1. Restricted Function Sets

The function set of genetic programming can be adapted to accommodate all kinds of different function symbols. The set of symbols that is necessary to solve a given problem is problem specific, so the function set should be adapted accordingly for each problem. A straight forward way to improve the comprehensibility of GP solutions is to use restricted function sets containing only simple functions. If a new problem is approached, different configurations with small and with large function sets should be tried in order to find out which function set is sufficient to find accurate models.

Generally it is not recommended to use very large function sets just because they are available in a given genetic programming implementation. Using a restricted function has the benefit that the final solutions are easier to interpret than with comprehensive function sets also including trigonometric functions, conditionals and boolean functions. Even though a simple function set containing only arithmetic operations (+, -, *, /) cannot express transcendental functions (e.g. $\exp(x)$) exactly it is possible for GP to find a good enough approximation [165]. Another benefit of arithmetic function sets is that models containing only arithmetic operators can be easily simplified symbolically

[146].

4.2.2. Automatic Simplification of Symbolic Expressions

The amount of unoptimized code in symbolic regression models is usually rather high, a large gain can often be achieved by arithmetic simplification of the mathematical expression. Automatic simplification should at least support folding of constants, aggregating sums and products, simplifying stacked fractions. These transformations are rather easy to implement and lead to often drastic reductions in solutions sizes. A set of algebraic simplification rules for symbolic regression are for instance given in [97, 88].

If more advanced function sets are used the simplification routine should be adapted accordingly. Function sets including conditionals and boolean expressions often lead to solutions with a lot of inviable code. Through simple transformations of conditionals and constant folding, all of the inviable code and a large part of the unoptimized code can be removed easily simply by removing branches that are not executed. This operation is a simple extension of arithmetic simplification [146].

4.2.3. Branch Impact Metrics

In Chapter 6 methods to calculate the impact of variables in final symbolic regression solutions are described. In the same way as the impact of variables was calculated, the impact of specific branches of symbolic regression models can be calculated.

The quality of the original model is used as a reference value. The impact of a branch can be calculated as the ratio of the quality of a manipulated model, where the branch is replaced by a constant value over the quality of the original model. Iterating over all branches of a symbolic expression tree the impact of all branches can be calculated in this fashion.

Multiple alternatives are possible to determine the constant value with which a branch should be replaced. We found that the median of the output values of this branch is a good choice for the constant value, as it is more robust to outliers than the arithmetic mean.

4.2.4. Visual Support for Manual Simplification

Branch impacts can be used effectively to guide domain experts in the analysis and simplification of symbolic regression models. A simple but effective way is to present the model in tree form and visually indicate the relevance of all branches. Branches that are more relevant have a strong coloring, while branches that have only weak impact are kept in the same color as the background. In the process of model simplification only information that is relevant for pruning should be presented to prevent visual clutter. Through the different coloring it is easy to determine quickly which parts of the model are relevant and which parts can be cut away easily. If a manual pruning tool is combined with online feedback of the model quality (for instance by an automatically updating scatter-plot or X-Y plot) this method can be very effective. Interactive manipulation of

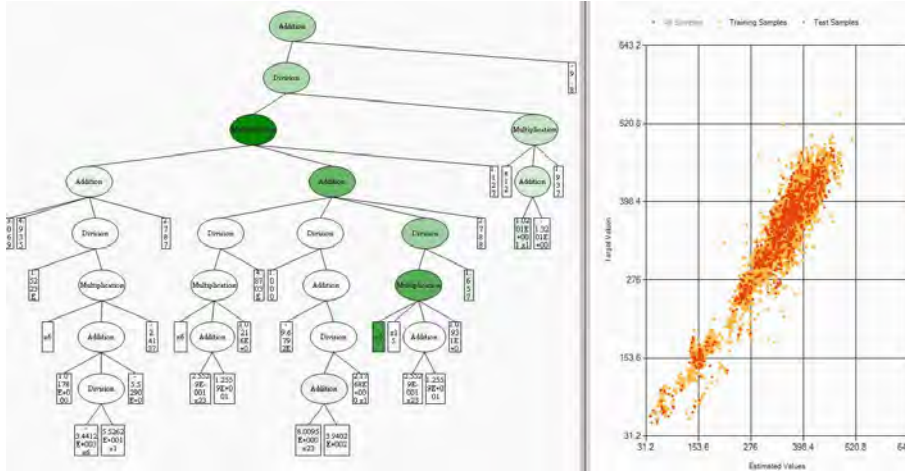


Figure 4.5.: Model simplification as implemented in HeuristicLab

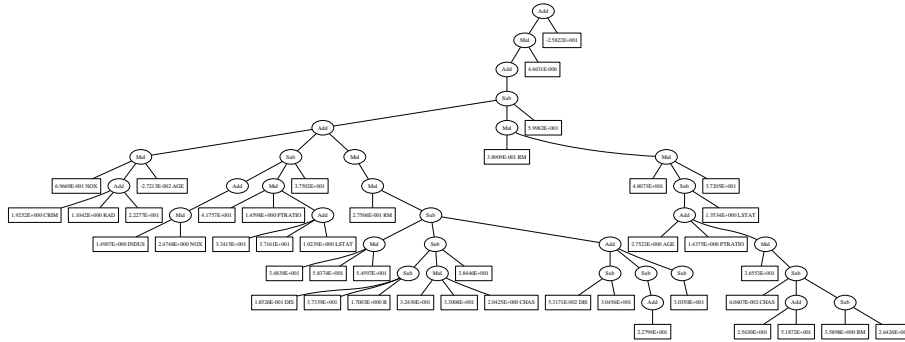


Figure 4.6.: Original symbolic regression solution as produced by GP for the Housing problem

the model can also leading to better understanding of the model and higher trust in the correctness of the model.

Figure 4.6 shows the original model produced by a GP run for the Housing dataset (see A.3.2). The model has a depth of twelve levels and a length of 76 nodes.

Figure 4.7 shows a semantically equal model which is the result of mathematical transformation of the original model shown in 4.6. In particular the outputs of the two models are equivalent. The depth is reduced to four levels and the length is reduced to 38 nodes. The models is already a lot more comprehensive. However the model still contains branches that have only a minor impact on the output. Such branches can be detected by calculating the branch impacts for each node. Using the branch impacts the model can be simplified further either manually or automatically.

Figure 4.8 shows the model after manual pruning of branches with low impact. The quality of this model is slightly worse but it is also a lot more compact and more com-

5. Generalization in Genetic Programming

Overfitting means to train, estimate or select models which do not generalize well to new observations. The effect can be observed when a model is applied to a dataset, which was not used in the training process and the fit on the new dataset is considerably worse, even though the model has a good fit on the training data. Overfitting occurs when a model is fit too tightly to the observations in the dataset in order to reduce the approximation error by increasing the complexity of the model. Observations in the training dataset are usually afflicted with a certain amount of noise. Thus, at a certain point the increased complexity is solely used to approximate random fluctuations in the dataset. This is counter-productive because at this point only the approximation error on the observations in the dataset can be reduced, and the expected error of the model for new observations increases. The model becomes useless for practical application. This leads to the problem of the well-known bias-variance trade-off in data-based modeling [83]

It is generally assumed that bloat and overfitting are related. However, it has been recently observed that overfitting can occur in absence of bloat, and vice versa. Thus, it has been suggested that overfitting and bloat are two separate phenomena in genetic programming [220], [190]. This suggests that overfitting and bloat should be controlled also by separate mechanisms.

One way to reduce overfitting is to use an internal validation set on which all solutions of the population are evaluated. A solution that has a comparable accuracy on the training and validation set is returned as the final result [73, 232, 231]. This approach can be extended to a solution archive in which Pareto optimal (quality and size) solutions on the validation set are kept [193, 180]. The result of the GP run is the set of solutions in the archive. The advantage of this approach is that an appropriate model can be selected a-posteriori because all good solutions on the validation set, discovered over the whole run, are available at the end. The problem of this approach is that without parsimony pressure the set of good solutions on the validation set is more likely to be extended at the end where the larger solutions are located. The chance that small solutions, which improve solutions which are already in the archive, are found in later stages of a GP run diminishes through the bloat-effect.

Other approaches to control overfitting and improve generalization that are often used in statistical learning methods are based either on the estimation of the expected generalization error or on penalization of overly complex models [83]. The first approach is to tune the algorithm parameters in iterative steps to find parameter settings which result in a model that generalizes well using an estimator for the expected generalization error of the model. The expected generalization error can be estimated using a hold-out set or through cross-validation. The second approach is to add a penalty term that

depends on the model complexity and optionally on the number of training samples to the objective function that is minimized. This approach integrates directly into the training algorithm and produces a model with a good balance of training error and complexity.

In practice the approach of using a validation set has turned out to be a working method to reduce the chance of choosing an overfit model [231]. The approach is simple and practical but not very efficient. If overfitting occurs in the early stages of the GP run a lot of effort is wasted evaluating solutions which are too complex, and often the solution found in the initial stages is not improved over the rest of the GP run because search concentrates on uninteresting areas of the search space.

A simple improvement is to introduce an early stopping criterion. If the best solution on the validation set is not updated for a predetermined number of generations, that GP run should be stopped. The basic idea is that starting a new GP run is more likely to produce a better solution on the validation sets than continuing the current GP run which is already in an overfitting stage. The value for the number of generations with no improvement after which the run should be stopped must be hand-tuned, based on the results observed in a few long running test runs. Setting this parameter can often be difficult, because if new genetic diversity is brought into the population through mutation or migration, the overfitting effect is reduced and a new validation-best solution can be found.

If overfitting can be detected reliably in the process, this information could be used effectively to adjust the complexity of the search space. First it is necessary to define a reliable measure for overfitting that can be calculated in a GP run. Then an effective scheme which acts as a complexity reducer, can be used which gradually moves the population into a non-overfitting solution space again. Gradual slow changes are preferred because a sudden change in complexity reduction for instance through exhaustive pruning of the whole population is likely to cause a significant loss in genetic diversity and might lead to premature convergence of the process.

The general setup should lead the process to search for solutions which perform equally well on the training and validation set. Contrary to the ambitions followed in 4 in this setup the maximal complexity of solutions is not constrained and there is generally no bias to search for compact solutions. Instead, as long as no overfitting occurs, the algorithm is allowed to generate more and more complex solutions. Most countermeasures against bloat discussed in the previous chapter can be combined with countermeasures against overfitting. Both phenomena should be treated by specific countermeasures. Bloat can be controlled through size limits or parsimony pressure, and overfitting can be reduced through validation and complexity reduction.

5.1. How to Detect Overfitting in GP

If an internal validation set is available then the validation set can be used to efficiently detect overfitting on the training set. Each solution candidate in a population is also evaluated on the validation set. So for each solution candidate two fitness values are

calculated, namely f_{training} and $f_{\text{validation}}$. Only the fitness on the training set is used for selection, so the fitness on the validation set can be used to detect when overfitting occurs. An indicator for overfitting is when the average or best validation fitness decreases, while the average or best training fitness is increased. A simple boolean function for overfitting at generation g based on this principle is 5.1. This metric returns true if the moving average of the average validation fitness $\text{MA}_k(\bar{f}_{\text{val}}, g)$ over the previous k generations at generation g decreases by more than a given ratio p over k generations.

$$\begin{aligned} \text{MA}_k(\bar{f}_{\text{val}}, g) &= \frac{1}{k} \sum_{i=1}^k \bar{f}_{\text{val}}(g-i) \\ \text{Threshold}_k(\bar{f}_{\text{val}}, g, p) &= (1-p)\bar{f}_{\text{val}}(g-k) \\ \text{Overfitting}(g)_{p,k} &= \begin{cases} \text{true} & \text{if } \text{MA}_k(\bar{f}_{\text{val}}, g) < \text{Threshold}_k(\bar{f}_{\text{val}}, g, p) \\ \text{false} & \text{otherwise} \end{cases} \end{aligned} \quad (5.1)$$

The problem with this overfitting detection function is that there is a rather large delay between the time when the algorithm actually starts to overfit and the time when the decrease in average fitness quality is detected. When overfitting is detected, it already caused a significant decrease in validation fitness, which means that the solution candidates in the whole population are already rather overfit. It is however not so easy to recognize overfitting earlier with this detection function as a minimal k is necessary to make the function robust against minor variations in average validation fitness.

Another issue with this function is that when it is used in combination with a complexity reduction method, the point at which overfitting does not occur anymore is difficult to detect, because the average validation quality also is decreased together with the average training quality when the complexity reduction mechanism is applied.

An alternative way to detect overfitting that is presented for the first time in this work is based on the correlation of training fitness and validation fitness of solution candidates in the population at generation g 5.2. Intuitively, if no overfitting occurs, the validation fitness of solution candidates should be strongly correlated to the training fitness. Solution candidates with larger training fitness are more likely to be selected for recombination, so ideally such solution candidates should also have a high validation fitness. Solution candidates that have low training fitness should also have low validation fitness. If the correlation of training and validation fitness is high, the selection pressure implicitly leads to solutions that have better training and validation fitness. In contrast, if the correlation of training and validation fitness is low, the selection pressure will lead to solution candidates that are better only in respect to training fitness, and the chance to create solutions with lower validation fitness increases.

In our experiments we used Spearman's rank correlation ρ [196] and an alpha value in the range 0.5 – 0.8.

$$\text{Overfitting}(g)_{\alpha} = \begin{cases} \text{true} & \text{if } \rho(f_{\text{training}}(g), f_{\text{validation}}(g)) < \alpha \\ \text{false} & \text{otherwise} \end{cases} \quad (5.2)$$

The correlation-based overfitting detection function indicates very early when the process starts to overfit. If the correlation value is low in the current generation the selection pressure causes a decrease in validation quality in the next generation. Because of this, any countermeasure for overfitting can be applied in early stages when the population does not yet contain a large number of overfit solution candidates.

A drawback of the correlation-based overfitting detection function is, that the fitness correlation is also low when under-fitting occurs, because in this situation the training fitness values are rather similar and a high training fitness is not necessarily related to a high validation fitness.

The fitness correlation is also low, if the population is converged to a small set of solution candidates with similar fitness values. In this situation the correlation is low because the spread of fitness values is small, thus small deviations in validation quality start to have a larger effect on the correlation value. Additionally, if the population is converged to a small set of solution candidates, the number of different fitness values is too small for the correlation coefficient to be significant.

The effect of any overfitting countermeasure can be measured with the same overfitting detection function. An effective countermeasure should lead to an increase in the fitness correlation value. Any overfitting countermeasure should not introduce strong biases or sudden changes in the search process, otherwise there is a risk of premature convergence or directing the process into under-fitting. If the interference is too strong the process could jump from an overfitting stage to under-fitting within just one generation and it would be impossible to detect that there was a state change with the correlation based overfitting detection function.

5.2. Countermeasures Against Overfitting

5.2.1. Restarts

If overfitting occurs in a run a pragmatic approach is to stop the current run and start another run. This assumes that a good solution has already been found and that it is unlikely to improve the current best solution by continuing the run which is already in an overfitting stage. Finally after a maximum number of restarts one of the final solutions is selected as the overall result. Selection of the final result should be based on overall fitness on training and validation set and ideally combine with a complexity metric to prefer more compact models.

Instead of starting a new completely independent run with the same algorithm configuration an adaptive approach could be more effective. Decreasing the maximally allowed program size or the maximal size of trees in the initial generation with each run should delay the overfitting stage. Alternatively gradually reducing the selection pressure with each restart for instance through reduction of the tournament group size should also delay the overfitting stage as the evolutionary process improves the training fitness more slowly.

5.2.2. Parsimony Pressure Against Overfitting

Overfitting can be reduced by reducing the complexity of solution candidates in the population. Intuitively, the complexity of solution candidates can be removed by reducing the average program length in the population. Parsimony pressure increases the probability of selection of smaller individuals by adjusting the fitness value of solutions to integrate the solution complexity. Covariant parsimony pressure [163] (also see previous Chapter 4) is an elegant way to dynamically adapt the parsimony pressure coefficient in order to tightly control the average program length over the GP run.

It has already been suggested by Poli and McPhee in [163] to dynamically turn covariant parsimony pressure on and off at specific stages of the GP run. This suggestion is taken up in this work to introduce a novel approach to reduce overfitting. If a robust overfitting detection function like the one explained in Section 5.1 is available, it is possible to use covariant parsimony pressure to gradually decrease the average program length to a point where no overfitting occurs. When the algorithm is back in a non-overfitting stage parsimony pressure can be turned off again. In this way the evolutionary process can freely increase the solution complexity to a level that is necessary to solve the problem and limit the complexity from above as soon as the process starts to overfit.

Again, the parsimony pressure should be configured in such a way to reduce the average program length only slowly. If parsimony pressure is applied too strongly there is the risk that the program length becomes the attribute that determines selection probability. This can happen for instance by forcing the average program size to a certain level which is a lot smaller than the current average program size. In such cases the adjusted fitness values are strongly correlated to the program length instead of program raw fitness and the evolutionary process will be mainly driven by the program length.

In the results section the results of the experiments with covariant parsimony pressure to counter overfitting are shown in Figure 5.3.

5.2.3. Pruning Against Overfitting

Pruning of GP solutions has been discussed already in the previous section. Pruning can also be used against overfitting. The idea is based on the hypothesis that the overfitting behavior is caused by small code fragments which have only a small positive impact on the training fitness but a large negative impact on the validation set and that these code fragments are spatially grouped. Such branches can be removed selectively by pruning operations. Pruning cannot be effective if the overfitting effect is caused by code fragments which are scattered over the whole program.

The effect of pruning is that code fragments which have no impact or only small impact on the program output are removed. The original intention of pruning is to create more compact final solutions or when used in the GP process to remove bloated code from the population in order to make room for effective code fragments. If overfitting is caused by small spatially grouped code fragments which only have a small impact on the program output pruning should remove exactly those code fragments and reduce

overfitting. Pruning operations which cause a slight deterioration in fitness should be possible and removal of smaller branches should be preferred.

If softening the distinction between the training and validation dataset partitions is allowed, the pruning operation can be changed to calculate the fitness deterioration on the validation fitness instead of the training fitness. This is meaningful for the application of pruning against overfitting because the branches responsible for bad validation fitness can be easily and robustly identified if the validation fitness is used to evaluate the effect of branch removal. Pruning operations can be applied very selectively only removing branches which actually decrease validation fitness. The drawback of this is that the validation dataset is used not only as an indicator for overfitting. Instead, the validation fitness has a direct impact on solution candidates in the population and so also has an equally strong effect on the evolutionary process as the training fitness. This leads to the risk that the process produces solutions which are overfit on the combination of training and validation fitness. This cannot be detected through observation of the trajectory of the fitness on the validation set.

Similarly to parsimony pressure, pruning can be integrated into the GP process as a step after evaluation of the current generation. If the overfitting detection function indicates that the process is in an overfitting stage all individuals of the current generation are pruned and evaluated. Only after this optional pruning step the next generation of the population is generated as usual.

Pruning reduces the genetic diversity in the population and should be applied only very carefully. Many different ways of pruning are possible [232], but in this work we only use a simple greedy pruning variant for symbolic expression tree encoding (see Algorithm 5). An effective pruning method should guarantee that the necessary building blocks of good solutions are kept unchanged and that code fragments are removed in such a way to make it possible to create new solution candidates of similar or better fitness.

In the previous section it has been shown that pruning can be used to control bloat. In this situation it often makes sense to prune only a part of the population, excluding the very best individuals. This is, however, not recommended if pruning is used against overfitting, because the best individuals of the population are more likely to be overfit. The best individuals also should be pruned. Simply pruning all individuals of the population without exceptions worked well in our experiments.

5.3. Experiments

To analyze and compare the effects of different anti-overfitting methods we used two datasets, on which SGP and OSGP produced overfit solutions. The first dataset is the Boston Housing dataset (see Section A.3.2). The second dataset is the Chemical-I dataset (see Section A.3.2). To show the overfitting behavior we evaluated all individuals of the population also on a test set in each generation. In this set of experiments the datasets have been partitioned into training-, validation- and hold-out set. Only the training set is used to calculate the fitness. The validation set is used to calculate the correlation of training- and validation fitness. The validation fitness is used by the algorithm only

Parameter	Value
Population size	2000
Max. generations	100
Parent selection	Proportional
Replacement	generational no elitism
Initialization	PTC2 [127]
Max. initial tree size	100
Crossover	Sub-tree swapping
Mutation	7% One-point, 7% sub-tree replacement
Model selection	Best on validation
Fitness function	R^2 (maximization)
Function set	+, -, *, /, average
Terminal set	constants, variables

Table 5.1.: Genetic programming parameters for the experiments.

to determine if overfitting occurs and thus if an overfitting countermeasure should be applied. The hold-out set is used to calculate the test fitness for reporting purposes only and is not used by the algorithm.

In all experiments the training-validation fitness correlation function is used to detect overfitting. The algorithm variants and shorthand symbols are standard GP without size constraints (SGP), SGP with static constraints (SGP-static), OSGP without size constraints (OSGP), OSGP with static constraints (OSGP-static), SGP with covariant parsimony pressure continuously over the whole run (SGP-CPP), SGP with conditional covariant parsimony pressure only in overfitting phases (SGP+adaptiveCPP), OSGP with conditional pruning only in overfitting phases (OSGP-pruning). The parameter settings common for all algorithm variants are shown in Table 5.1

The algorithm for overfitting detection that is used in algorithms SGP-pruning, OSGP-pruning, SGP-adaptiveCPP, and OSGP-adaptiveCPP is shown in Algorithm 6. A boolean variable *is-overfitting* is introduced which is initially set to false. After each iteration the training- and validation fitness correlation is calculated and the *is-overfitting* flag is updated accordingly. To prevent unstable behavior two threshold values $\text{thresh}_{\text{low}}$ and $\text{thresh}_{\text{high}}$ are used to toggle the value of the *is-overfitting* variable. In the experiments we used $\text{thresh}_{\text{low}} = 0.35$, $\text{thresh}_{\text{high}} = 0.65$.

Algorithm 6: Algorithm for overfitting detection.

```

 $r \leftarrow \rho(f_{\text{training}}(g), f_{\text{validation}}(g));$ 
if is-overfitting = false  $\wedge r < \text{thresh}_{\text{low}}$  then
  | is-overfitting  $\leftarrow$  true;
else if is-overfitting = false  $\wedge r < \text{thresh}_{\text{high}}$  then
  | is-overfitting  $\leftarrow$  false;
end

```

In SGP-adaptiveCPP the covariant parsimony pressure is adapted based on the *is-overfitting* flag as shown in Equation 5.3. When the algorithm is in an overfitting state the covariant parsimony pressure is adapted to reduce the average program length by five percent each iteration. In contrast, in a non-overfitting state “negative” parsimony pressure is applied to increase the average program length by five percent in each iteration.

$$\begin{aligned}
f_{\text{adjusted}}(x) &= f(x) - c(t)\ell(x) \\
c(t) &= \frac{\text{Cov}(\ell, f) - \delta_\mu \bar{f}}{\text{Var}(\ell) - \delta_\mu \bar{\ell}} \\
\delta_\mu &= \begin{cases} -0.05 \bar{\ell} & \text{if is-overfitting} = \text{true} \\ +0.05 \bar{\ell} & \text{if is-overfitting} = \text{false} \end{cases}
\end{aligned} \tag{5.3}$$

5.4. Results

In the following Figures 5.2, 5.3, 5.4, and 5.5 the results of experiments with different genetic programming configurations are shown. The result plots show trajectories of the best fitness on the test set and of the training-validation fitness correlation over the whole run. Fitness function is the squared correlation coefficient (R^2) which is applied on the estimated and target values. The values shown in the charts are median values over thirty independent runs for each configuration.

5.4.1. Results: Covariant Parsimony Pressure

Figure 5.1 shows the dynamics of a typical SGP run with the housing dataset. The line chart in the top left shows the trajectories of the best and average fitness on the training partition and on the test partition over 100 generations. The chart shows that the fitness on the training partition steadily increases, while the fitness on the test set decreases at the later stages of the GP run. This demonstrates clearly that overfitting occurs. The top right line chart shows the trajectory of the correlation of training- and validation fitness for this run. The correlation decreases at a very early point in this run and it can be observed that the turning point is at the generation where the average test fitness also starts to decrease. So the correlation is a good indicator for overfitting.

The four panels in the bottom of Figure 5.1 show scatter plots of the training and validation fitness of all models in the population at specific generations in the exemplary run. In the first generation the correlation is rather low, the maximum correlation is reached in generation eight. After generation eight the correlation decreases and it can be observed in the scatter plots that the number of individuals with high training fitness but low validation fitness increases.

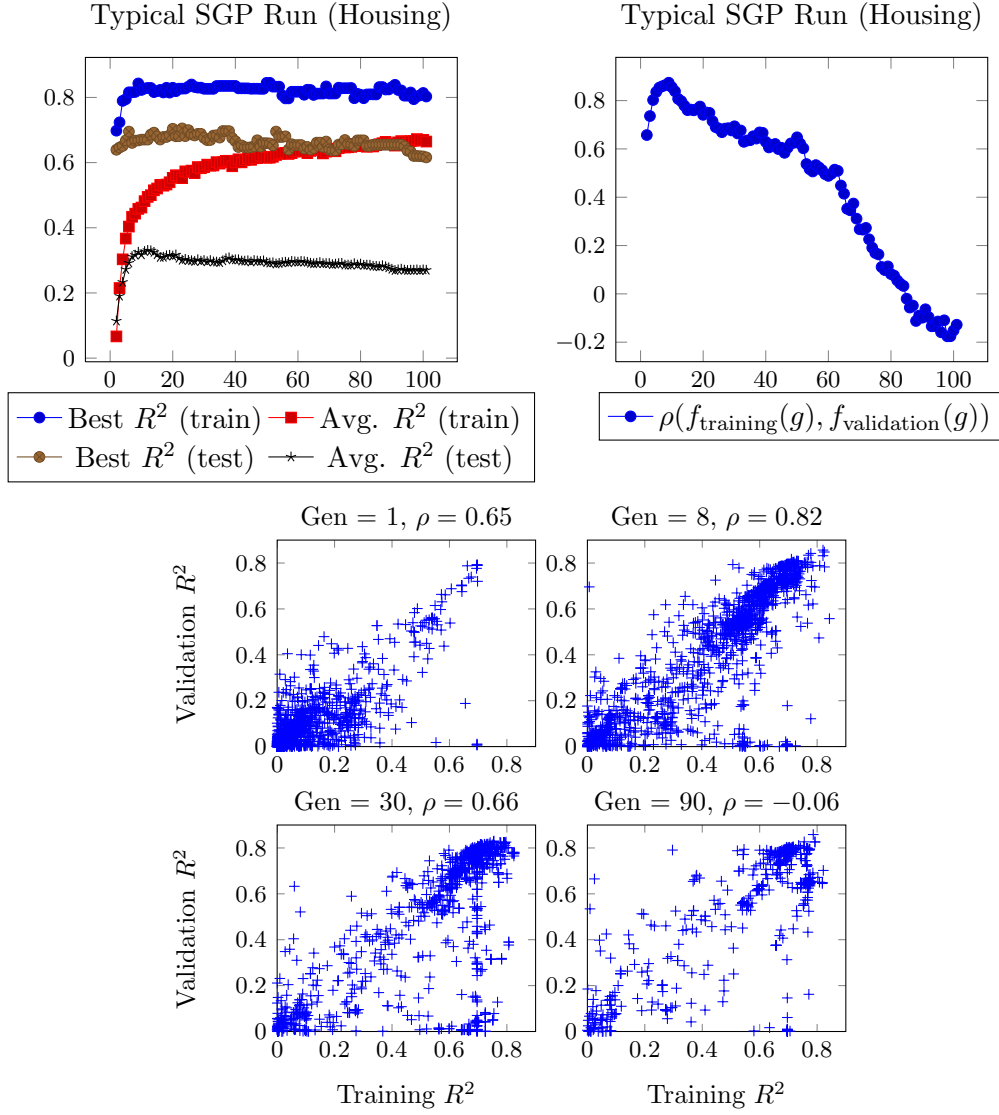


Figure 5.1.: Trajectories of training and test fitness and $\rho(f_{\text{training}}(g), f_{\text{validation}}(g))$ over 100 generations in an exemplary run of SGP for the housing dataset. The scatter plots in the lower half show the training vs. validation fitness of all models in the population at four different generations.

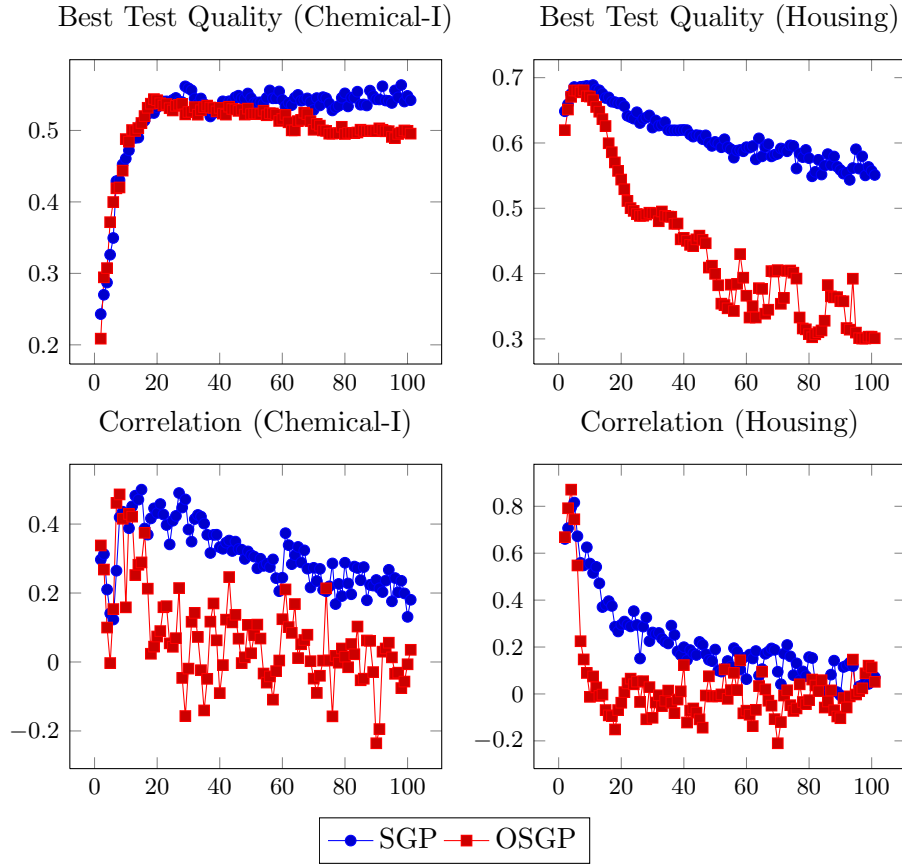


Figure 5.2.: Overfitting behavior of SGP and OSGP without size limits and without overfitting control.

5.4.2. Results: SGP and OSGP

Figure 5.2 shows the best test quality and the correlation between training and validation quality for standard GP and GP with offspring selection for two different problems. In this first set of experiments no limits have been set for either program length or depth. For the Chemical-I dataset SGP does not overfit but with offspring selection overfitting occurs in the late stage of the run. For the Housing dataset both algorithms overfit. The correlation between training and validation quality is low for both problems and for both algorithms. OSGP has a lower correlation value in both cases.

Figure 5.3 shows the results of SGP and OSGP with static limits and SGP with covariant parsimony pressure. For SGP-static and OSGP-static the static limit program length is 250 nodes and the depth limit is 17 levels. SGP-CPP is configured to keep the average program length at 100 nodes over the whole run through covariant parsimony pressure. OSGP-static overfits for both problems. SGP-static overfits only on the Housing problem. SGP-CPP works best but still overfits on the Housing problem.

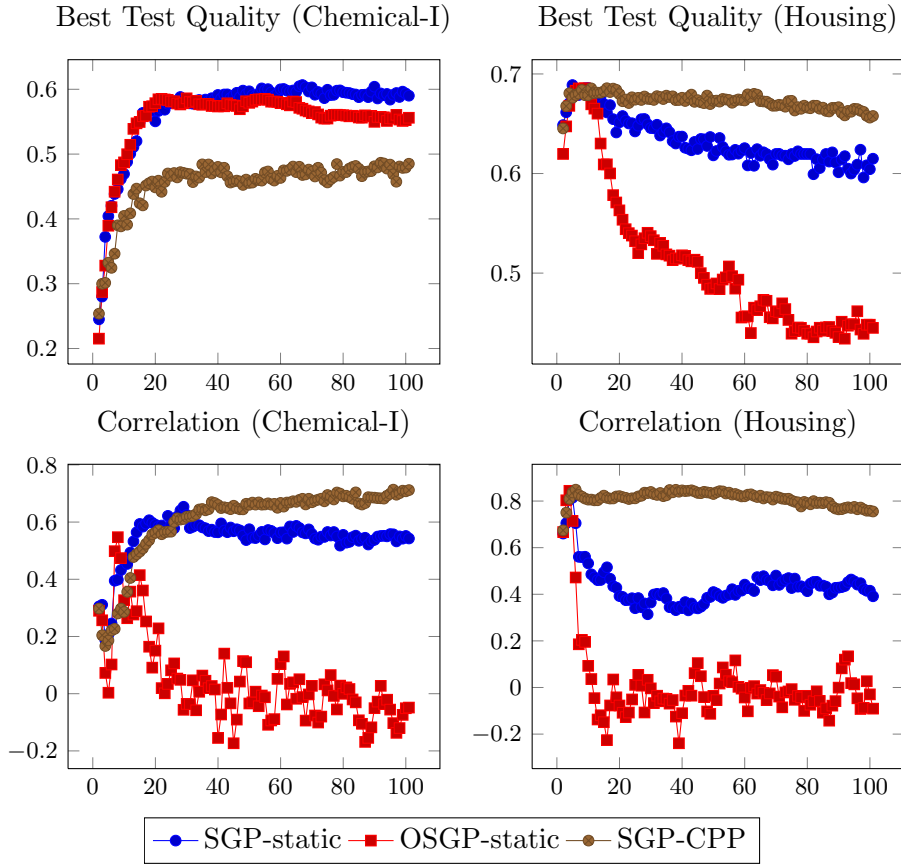


Figure 5.3.: Overfitting behavior of SGP-static, OSGP-static, and SGP-CPP without overfitting control.

The correlation of training and validation fitness is again very low for OSGP-static but relatively high for SGP-static and SGP-CPP.

5.4.3. Results: Adaptive CPP and Overfitting-triggered Pruning

Figure 5.4 shows the best test quality and training-validation correlation for SGP with overfitting detection and adaptive covariant parsimony pressure and SGP with overfitting-triggered pruning. For comparison the values for SGP without size constraints are also shown in the same plots. For the Chemical-I problem the target range for the correlation value was 0.45 – 0.65. For the Housing problem a range of 0.45 – 0.75 was used. The plots of the training-validation correlation show that both methods, pruning and adaptive parsimony pressure successfully steered the algorithm to keep the correlation in the target range. For the Chemical-I problem no overfitting can be observed and both variants lead to better solution quality on the test set. For the Housing problem minor overfitting can be observed but both variants are better than

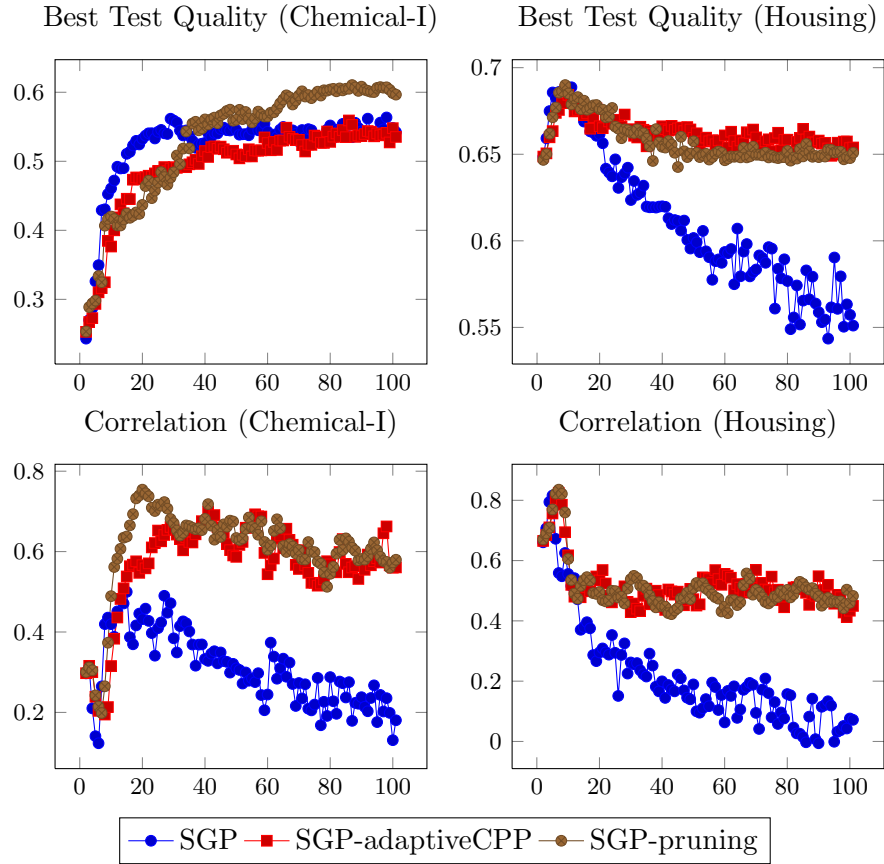


Figure 5.4.: Overfitting behavior of SGP, SGP adaptive covariant parsimony pressure, and SGP with overfitting-triggered pruning.

SGP.

Figure 5.5 shows the best test quality and training-validation fitness correlation for OSGP with static size limits and OSGP with pruning as overfitting countermeasure. Pruning was applied only when the training-validation fitness correlation indicated that overfitting occurred. The target range for ρ was set to 0.45 – 0.75 for the Housing problem and 0.45 – 0.65 for the Chemical-I problem. For both problems pruning was applied to the whole population (two iterations with a maximum pruning ratio of 50%, tournament group size 100). The results show that in the case of the Chemical-I problem pruning effectively removes overfitting and that the training-validation fitness correlation is in the target range over the whole run. In the housing problem OSGP with pruning does not overfit so strongly as OSGP with only size limits, but overfitting still occurs as can be seen by the low training-validation fitness correlation value. Even with pruning the correlation value drops below the target range.

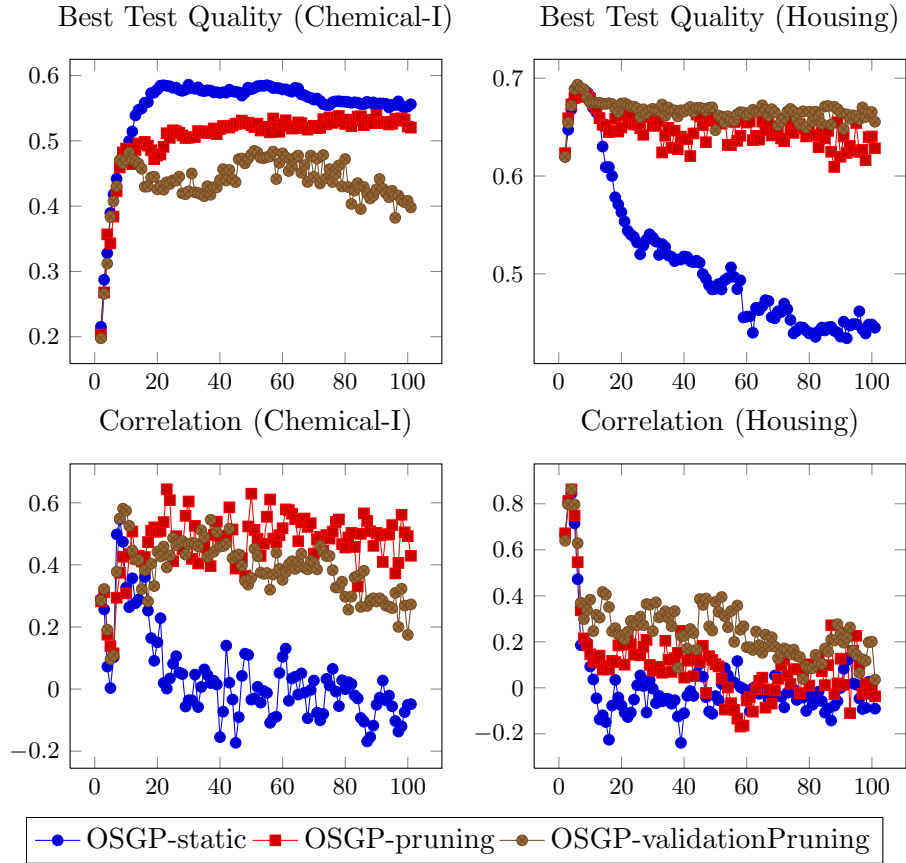


Figure 5.5.: Overfitting behavior of OSGP-static, OSGP with overfitting-triggered pruning, and OSGP with overfitting-triggered pruning based on the validation set.

Algorithm	Size		R^2 (train)		R^2 (test)	
OSGP	1,805.5	± 268.5	0.45	± 0.09	0.39	± 0.07
OSGP-static	247.5	± 4.1	0.44	± 0.08	0.40	± 0.08
OSGP-adaptivePruning	31.9	± 4.7	0.55	± 0.05	0.47	± 0.04
OSGP-adaptiveValPruning	25.5	± 2.8	0.41	± 0.04	0.34	± 0.04
SGP	1,542.8	± 334.8	0.54	± 0.06	0.43	± 0.05
SGP-static	205.1	± 9.8	0.56	± 0.06	0.44	± 0.06
SGP-staticCPP	31.0	± 5.0	0.45	± 0.04	0.36	± 0.04
SGP-adaptiveCPP	96.9	± 32.4	0.53	± 0.05	0.45	± 0.04
SGP-adaptivePruning	105.2	± 19.0	0.60	± 0.03	0.51	± 0.04
SGP-adaptiveValPruning	107.0	± 26.4	0.29	± 0.09	0.23	± 0.08

Table 5.2.: Summary of best training solution results of overfitting experiments for the Chemical-I problem (confidence intervals for $\alpha = 0.05$).

Summary of Overfitting Results

In Tables 5.2 and 5.3 the results of all overfitting experiments are summarized. The tables show average values over thirty independent GP runs for each configuration. Additionally the confidence interval for $\alpha = 0.05$ is given assuming normally distributed values. In the size column the length of the final solution is given. The columns R^2 (train) and R^2 (test) give the fitness of the final solution on the training and test set. The final solution is the solution with the best fitness on the training set over the whole run.

Table 5.2 shows that SGP with overfitting-triggered pruning (SGP-adaptivePruning) produced the best solutions on the test set (and on the training set). The worst results on the test set have been produced by OSGP with overfitting-triggered pruning based on validation fitness (OSGP-adaptiveValPruning). The runs with OSGP and pruning produced very small solutions with low fitness values. Static limits have only been for SGP-static OSGP-static, but the program size is also small for the configurations with overfitting control methods. This means that pruning and adaptive-CPP also prevented bloat.

Table 5.3 shows that the best results on the test set for the Housing dataset have been produced by OSGP with static covariant parsimony pressure. The solutions produced by this configuration are also all relatively small. The worst results on the test set have been produced by SGP with static size constraints.

Algorithm	Size		R^2 (train)		R^2 (test)	
OSGP	2,604.4	± 431.4	0.56	± 0.09	0.30	± 0.06
OSGP-static	250.4	± 1.1	0.62	± 0.08	0.29	± 0.06
OSGP-staticCPP	34.4	± 6.1	0.77	± 0.04	0.42	± 0.05
OSGP-adaptivePruning	25.9	± 2.4	0.76	± 0.02	0.38	± 0.05
OSGP-adaptiveValPruning	28.5	± 3.0	0.78	± 0.02	0.39	± 0.06
SGP	2,207.0	± 364.9	0.45	± 0.11	0.30	± 0.07
SGP-static	226.6	± 10.2	0.61	± 0.09	0.28	± 0.05
SGP-adaptiveCPP	83.3	± 27.3	0.76	± 0.02	0.35	± 0.04
SGP-adaptivePruning	64.0	± 21.0	0.72	± 0.02	0.32	± 0.05
SGP-adaptiveValPruning	68.0	± 20.6	0.74	± 0.03	0.33	± 0.05

Table 5.3.: Summary of best training solution results of overfitting experiments for the Housing problem (confidence intervals for $\alpha = 0.05$).

6. Genetic Programming and Data Mining

6.1. Genetic Programming

Genetic programming as a general problem solving meta-heuristic is especially suitable for data mining tasks. The foremost goal of data mining is to find interesting patterns in large datasets which lead to the discovery of knowledge. The most important property of GP in the data mining aspect is that it produces interpretable white box models. This is an important factor to make knowledge discovery from GP models feasible. Another essential property of GP in the data mining aspect is that it simultaneously evolves the structure of the model and the parameters of the model. When mining data, the necessary complexity and structure of models is usually not known a priori. To the contrary the structure and complexity of the model should be a result of data mining, leading to knowledge about the inherent complexity in the data.

6.1.1. Flexible Model Representation

Genetic programming can be adapted easily to search for different kinds of patterns. This work concentrates mainly on regression and time series forecasting. In these data mining tasks, patterns are usually symbolic functional expressions which relate the input variables to the target variable. For binary classification tasks the pattern is often a discriminant function, which is again just a symbolic functional expression. Class labels are generated by wrapping the discriminant function into a function which determines the class label based on the output of the discriminant function and a class separator value. For classification tasks another commonly used pattern structure is a set of conditional if-then rules. This structure is often preferred because it is more interpretable [70]. Enforcing that the structure of the GP solutions is a set of if-then rules is different than simply adding conditionals to the function set. Conditional functions can also be useful in regression models, but usually the models are not restricted to a set of if-then rules. Genetic programming can be adapted to evolve all kinds of different patterns by configuration of the symbol set. In grammar-based GP systems the pattern structure can be specified more rigidly by defining a grammar for pattern expressions. In strongly-typed GP systems the pattern structure can be restricted by specifying type-constraints on symbols.

6.1.2. Implicit Feature Selection

A frequent task in data mining is the identification of relevant variables or feature selection. Usually, a large set of variables is available to describe a given fact, however, it

is assumed that only a subset of these variables is actually relevant. Determining this subset of relevant variables has several advantages. First of all information about the relevant variables is valuable in itself. Although there are no details about the relation of the input variables to the target variable (e.g. positive or negative effect), the set of important variables is easy to understand and can already increase knowledge considerably. Additionally, in subsequent modeling steps one can concentrate on explaining effects of the relevant variables on the target variable in detail, and the resulting models are easier to understand and less prone to overfitting.

Determining the subset of relevant variables is usually non-trivial, especially when there are non-linear or conditional relations. Implicit dependencies in the variables also make this task more difficult, as this ultimately leads to multiple sets of different variables which are equally plausible.

Feature selection is absolutely necessary for high-dimensional datasets where the number of variables exceeds the number of observations ($p \gg N$). Such problems have become increasingly important lately and frequently occur in bio-informatics. For instance in the analysis of micro-array datasets the number of observations is often small (around 50), however, one observation includes measurements of many thousands of variables. In such situations it is often possible to find correlated variables by pure chance, even though the variables are actually independent. Thus, overfitting and feature selection are important topics in such applications. The models are often strongly regularized in order to avoid overfitting. Additionally, it is necessary to rigorously assert the validity of models and training algorithms through methods like cross-validation and hold-out sets, and by calculating confidence intervals and statistical significance for results. The intricacies of high-dimensional problems have spurred the development of various specialized data-analysis algorithms (cf. [83]). In this work this topic is not further discussed, instead we concentrate on data-analysis problems where the number of observations exceeds the number of variables ($N \gg p$). Such problems frequently occur in the analysis of technical systems.

In genetic programming feature selection is implicit because fitness-based selection has the effect, that models, which contain relevant variables, are more likely to be included in the next generation. Over many generations this causes a genetic drift so that references to relevant variables are more frequent than references to irrelevant variables. It has been shown that GP is even applicable for feature selection problems with many variables but only few observations [114]. The implicit feature selection of genetic programming also removes variables which are pairwise strongly correlated but irrelevant for the target variable. However, if correlated variables also affect the target variable value, GP does not recognize that one of the variables can be removed, and it keeps both correlated variables.

6.1.3. Non-Determinism

While GP has some advantages that make it an ideal method for data mining there are also some obvious disadvantages that need to be addressed appropriately. One disadvantage of GP is that it is a non-deterministic method. Solving the same problem with

the same parameter settings multiple times with GP produces multiple different models. Even though GP is a global search method there is no guarantee that a single GP run will produce a result that is at a global optimum. The final solutions of different runs have a different output response but, which is even more crucial, also often have a different structure. The virtually infinite solution space of genetic programming combined with the non-deterministic search are the main reasons for this effect. Such effects are magnified, if the dataset contains variables which are interrelated. If this is the case GP will produce diverse solutions, using different sets of variables to model the same response of the underlying system. All this leads to the problem that, if the solutions of multiple GP runs are analyzed they are not only different in accuracy but also in structure, which input variables are used and which input variables have a strong effect on the model output. This reduces the trust in the models and the method, and the knowledge that can be gained from such a set of solutions is rather limited.

The power to find diverse solutions of good quality, if multiple equivalent solutions are possible, can also be seen as a virtue of genetic programming in comparison to deterministic methods which produce the same model each time even if alternative formulations are possible. The problem is rather to extract the interesting information out of a large solution set produced by GP. A method is needed which analyzes the solution set and extracts which parts are common to many solutions and which parts can be used interchangeably because alternative formulations are possible.

6.1.4. Training Performance

Another disadvantage of GP is that it is rather slow compared to other non-linear data mining methods. However, this is often not an issue, as the time spent for data mining is only a small part of the time spent in the whole process of knowledge discovery from databases. The data mining task can easily take a few days; however, the analysis of results produced in this step often takes much longer. Genetic programming and all other methods from the family of evolutionary algorithms additionally have the advantage, that they are embarrassingly easy to parallelize, as the population-based concept allows independent and potentially concurrent evaluation of solutions.

In any case, the factor that as many interesting patterns as possible should be uncovered is more important than the time spent to find the patterns. The fact that GP produces white box models and interpretable results reduces the effort that is necessary in the result analysis step.

6.1.5. Predictive Accuracy

An advantage of symbolic regression is that the resulting models are simple mathematical expressions. Thus, it is possible to analyze the structure of the model and the variable interrelations expressed in the model in detail. This is often helpful for model validation or to gain knowledge about previously unknown variable interrelations. The drawback is that simple models usually produce less accurate approximations than more complex models, as for instance produced by support vector regression. A good trade-off that

balances the predictive accuracy and the comprehensibility of the model is necessary.

In practical applications it is recommended to explore the full space of possible models, including highly accurate but complex models and only slightly accurate but very simple models. We observed that SVM models often have a very high predictive accuracy and linear regression can be used to establish a base line for the accuracy. Symbolic regression models produced with genetic programming often lie in the middle between SVM and LR models. A successful approach should combine the information gathered through different modeling algorithms. Usually, the symbolic regression model only includes a small subset of all available variables. Thus, the accuracy of a symbolic regression model should also be compared to the accuracy of SVM and LR models using the same subset of input variables. Such comparisons should also include residual analysis, because it quickly shows if a LR model is not sufficient and non-linear models are necessary. Ideally, a symbolic regression model should have an accuracy comparable to the accuracy of a SVM model using the same set of input variables. If the LR model also has a comparable accuracy and the residual analysis indicates no problem then the LR model should be preferred over both the SVM and symbolic regression model.

In this thesis the accuracy of a comparable SVM model for a symbolic regression model is only explicitly stated for one example (see Section 6.5.1). In this example the accuracy of the SVM model is significantly better than the accuracy achieved by the best symbolic regression model. However, in this thesis we explicitly do not strive to find the best possible regression models, instead the major concern is comprehensibility.

6.2. Analysis of Relevant Variables

Knowledge about the minimal set of input variables, that are necessary to describe a given system response, is often very valuable for domain experts. This information can improve the overall understanding of the examined system as it is easy to understand and verify.

6.2.1. Relation to Feature Selection

Feature selection methods are a way to determine the subset of relevant variables for a model. Especially when a large number of features are available compared to the number of samples it is necessary to reduce the set of features used in the model, in order make models interpretable and to prevent problems with overfitting. A survey of feature selection methods is given in [78]. Feature selection is often used as a wrapper around the model building step, where the subset of features is iteratively adapted until a model with acceptable balance between accuracy and complexity is found. Two general variants of this kind are forward and backward selection. Forward selection increases the size of the feature set starting from an empty set. Backward selection works in the other direction, reducing the size of the feature set in each step. Usually it is not feasible to try each possible combination of features because of the large number of possible combinations. So forward and backward selection are often greedy using heuristic functions to determine the utility of a given feature. The implication of this is,

that feature selection methods are not guaranteed in the general case to find the optimal subset of features for a given maximal size.

Feature selection methods can help to reduce the set of features to a subset of features, which are apparently relevant to model the unknown response function. Often it is also interesting to calculate variable ranks or weights indicating the relative importance of the features. Variable ranking is a by-product of iterative feature selection methods, as the order of adding or removing variables is determined by a heuristic relevance criterion. In general variable ranking and relative variable importance is problematic because of non-linear relations between features. See [78] for a discussion of such problems.

6.2.2. Relative Variable Importance in Linear Models

For linear regression the set of necessary variables can be determined through variable selection or via shrinkage methods [83]. Variable selection methods usually wrap linear regression and iteratively create new models adapting the subset of variables at each step until a satisfying solution is found. Shrinkage methods integrate variable selection directly into the modeling step by adding penalty terms for the number of variables into the cost function. Ridge regression and the Lasso method are examples for shrinkage methods for linear regression [83].

Linear regression models provide a set of necessary variables and also include information about the relative importance of the variables. The variable coefficient immediately indicates if the variable is proportional or indirectly proportional to the target variable.

Even for linear models it is not straight forward to determine the relative importance of variables. The issue of relative variable importance for linear methods has been discussed intensively. Surveys of the main contributions regarding this topic are given in [87, 64, 109].

Early approaches based on decompositions of different accuracy metrics lead to inconsistent results if the regressors are correlated. This issue has been addressed later in [122], where an approach based on the average reduction of residual variance over all orderings of regressors has been proposed, which was later also reinvented by [108]. The purpose of averaging over all orderings of regressors is to get more robust importance values in situations with correlated regressors. This approach has been extended in [214, 215] based on information-theoretic error measures.

In a recent contribution [77] the approach of [122, 108] is revisited and analyzed through simulation studies. Another recent contribution proposes the proportional marginal decomposition method [63] to calculate relative variable importance.

6.2.3. Variable Importance in GP

As stated in the beginning of this chapter genetic programming can be used for the analysis of relevant variables. Depending on the set of allowed symbols linear, non-linear and conditional impact factors can be identified reliably. This section discusses a method to gather information about the relevant variables in a system through multiple independent GP runs.

Two variants to approximate the relevance of variables for genetic programming have been described in [232]. The first approach is based on the frequency of variables in the population. The second approach is based on a variable impact measure over all models of the population. Even though the metrics have been formulated primarily to estimate population diversity they can also be used to calculate estimates for the relevance of variables on the response function of a single model.

Frequency-based Variable Importance

In the frequency-based approach variable impact is either the sum of variable references in all models or the number of models referencing a variable in the population. Both variants of frequency-based impact estimation have minor drawbacks. The first variant does not take the problem of code growth into consideration. The impacts calculated for two different generations cannot be compared directly because the average size of the solutions in the generations are different and thus the number of variable references are also different. This issue can be resolved by calculating a relative number of variable references. The drawback of the second variant is, that the metric does not account for multiple references to the same variable in the same model.

Impact-based Variable Importance

For the impact-based variable relevance metric the impact of each variable over all models of the population is calculated. To calculate the impact of a variable in a model, the difference of the response of the model on the original and a manipulated dataset is calculated. The idea is to manipulate the dataset in such a way as to remove the information of the variable for which the impact should be calculated. If the difference of the two responses on the original and the manipulated dataset is large this means, that the values of the variable had a big influence on the response of the model and the impact of the variable is large.

Critique of Variable Impact Calculation Through Replacement With Means

One approach to calculate the impact of a variable x_i in a model $m(x_1, \dots, x_i, \dots, x_k)$ is based on the increase of the model error that is occurred by replacing x_i by its mean value. The integral error $I_\Delta(f, m)$ of a model for a given distance function $\Delta(x, y)$ and an optional weighting function w is:

$$I_\Delta(f, m) = \int_D w(\mathbf{x}) \Delta(f(\mathbf{x}), m(\mathbf{x})) d\mathbf{x}$$

Usually the function f and the distribution of \mathbf{x} is unknown so the expected error $E_\Delta(y, m)$ over a set of n samples $Z_n = (\mathbf{x}, y)_n$ is used as an approximation.

$$E_\Delta(Z_n, m) = \frac{1}{n} \sum_{(\mathbf{x}, y) \in Z_n} \Delta(y, m(\mathbf{x}))$$

$$h_1(x, y) = \frac{4}{\sqrt{\pi}} * \exp(-16x^2) * \frac{2}{\sqrt{\pi}} * \exp(-4y^2)$$

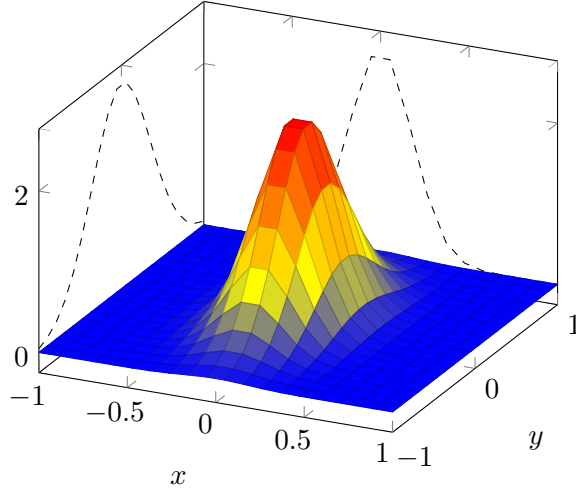


Figure 6.1.: Artificial function $h_1(x, y)$ and its response surface

The impact of variable x_i is the difference of the expected error of the original model response and the model response when x_i is replaced by its mean $m(x_1, \dots, \bar{x}_i, \dots, x_k)$. A large increase in the distance between model response and function response indicates that x_i has a large impact on the output of the model.

$$Impact(x_i, Z_n, m) = E_{\Delta}(Z_n, m) - E_{\Delta}(Z'_n, m), Z'_n = (x_1, \dots, \bar{x}_i, \dots, x_k, y)_n$$

This way of impact calculation is problematic and can lead to misleading impact values even for very simple functions. This is mainly caused by the fact, that a single constant value is used as a replacement value and the impact is calculated as a distance of the original response and the response of the manipulated model.

We use the following simple example of a bivariate function to demonstrate the problems of this approach. Figure 6.1 shows a bell-shaped function $h_1(x, y)$ with a larger gradient in the x direction than in the y direction.

The impacts of x and y should be determined for this example function h_1 by replacing the values by their mean values. For this purpose we assume the variables x and y are uniformly distributed in the range $[-1, 1]$. In this example the distribution of x and the function f are known, so we can calculate the integral errors exactly. Figure 6.2 shows the response surfaces of the squared differences $\Delta_{h_1, x}(x, y)$, $\Delta_{h_1, y}(x, y)$ when x and y are replaced by $\bar{x} = 0$ and $\bar{y} = 0$. The volume under the response surfaces represents the impacts of x and y . The integral of $\Delta_{h_1, x}(x, y)$ is 5.80 and the integral of $\Delta_{h_1, y}(x, y)$ is 1.75, so by this definition of variable impact the impact of x is more than three times larger than the impact of y .

It is trivial to approximate the impacts of x and y using a sample $Z_n = (x, y, f(x, y))_n$

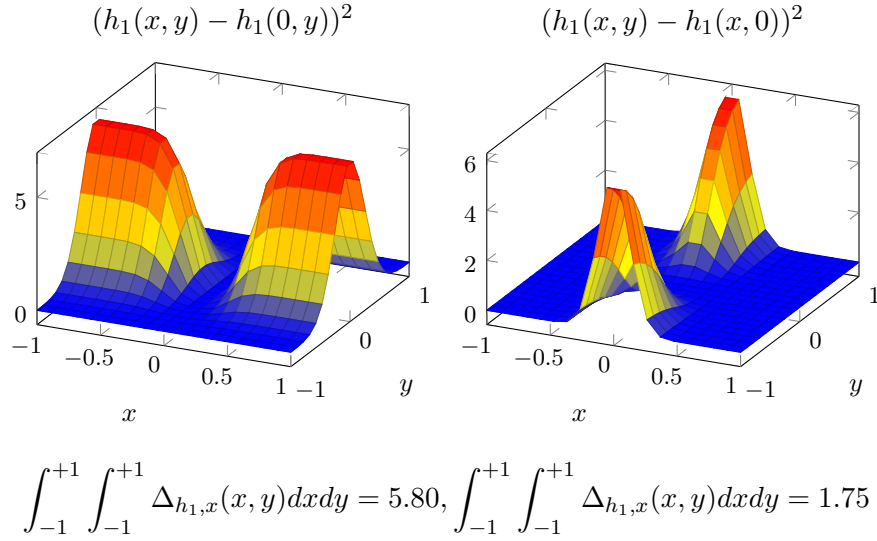


Figure 6.2.: Squared difference functions $\Delta_{h_1,x}(x, y)$ and $\Delta_{h_1,y}(x, y)$.

and the expected error. This calculation can be generalized for other distributions of x and y , using a weighting function $w(x)$ in the integral error function.

To show that the impact values calculated for h_1 are not representative consider the function $h_2(x, y)$ which is a transformation of h_1 . Figure 6.3 shows the function and its response surface. The difference of h_2 to h_1 is that the location of the maximum value is changed. The impacts of x and y on h_2 should be nearly the same as before, because the only difference is the location of the function.

Again we calculate the impacts of x and y using the same method as before. Figure 6.4 shows the squared difference functions $\Delta_{h_2,x}(x, y)$ and $\Delta_{h_2,y}(x, y)$. In this case the integral of $\Delta_{h_2,x}(x, y)$ is 1.27 and the integral of $\Delta_{h_2,y}(x, y)$ is 1.75. Because of the shift of maximum of the response function the impact of x on h_2 is a lot smaller than on the original function h_1 , the impact of y is the same for both functions. For h_2 the impact of x is even smaller than the impact of y .

The different impact results stem from the fact that $h_1(E[x], E[y])$ and $h_2(E[x], E[y])$ are different, so different replacement values are used, leading to different squared distances. The replacement value has a strong effect on the impact calculation, even though h_1 and h_2 are similar, the different replacement values lead to very different variable impacts. If the expected error is used to calculate the variable impact of x_i the replacement value \bar{x}_i depends on the available samples of x_i and the impact calculation is even more unstable.

In [232] an alternative way to calculate the variable impact is proposed that is based on noising the variable values by adding normally distributed values to the original variable values. This approach does not suffer the problems outlined in this section and should be preferred. The disadvantages of this method are that it depends on the parameter

$$h_2(x, y) = \left(\frac{4}{\sqrt{\pi}} * \exp(-16(x-1)^2) + \frac{4}{\sqrt{\pi}} * \exp(-16(x+1)^2) \right) * \frac{2}{\sqrt{\pi}} * \exp(-4y^2)$$

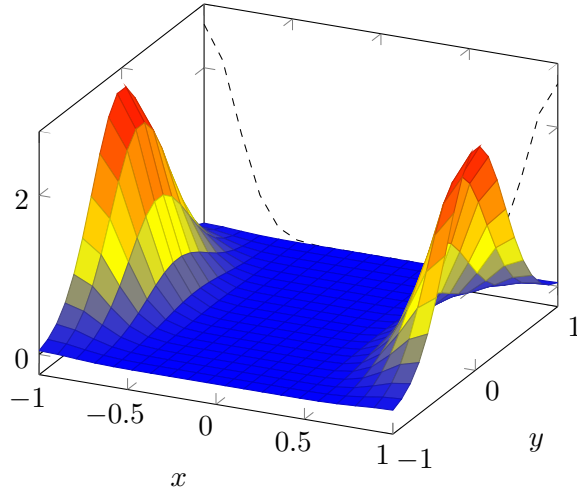


Figure 6.3.: Artificial function $h_2(x, y)$ and its response surface

σ which determines the variance of the additive noise term and that the distribution of the input values is changed if the original values are not normally distributed.

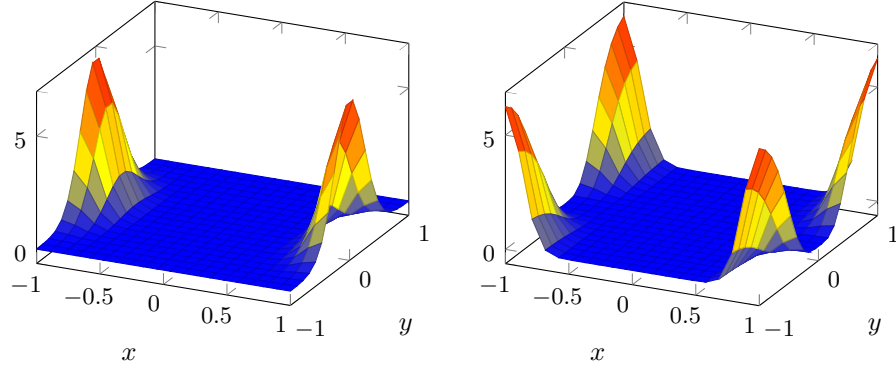
6.2.4. Variable Importance in Random Forests

Random forests are a supervised learning approach for classification and regression proposed by [26]. The approach is based on the observation that ensembles of classification or regression models can produce better estimations than a single model [25]. The estimated value for an ensemble of models is calculated by a voting scheme or by averaging over the estimated values of all models. This has the effect that it is less likely to receive estimations with a large error because of the averaging effect over many models. The requirement for this positive effect is, that the models in the ensemble should be structurally diverse. Ensemble methods additionally have the nice property that estimates for the generalization error can often be calculated easily without cross-validation [24, 26] and that confidence intervals for estimated values can be calculated naturally.

Each model in a random forests is a classification and regression tree (CART) [28] trained on a different subset of training samples and deliberately not pruned which is usually necessary for CART to prevent overfitting. Even though the random forest can contain overfit models the ensemble is not likely to overfit because a few outliers do not have a large impact on the average prediction value over all models.

The disadvantage of random forests is that the unique property of CART that the models are very easy to interpret and analyze is lost because of the large number of structurally diverse trees in the forest. Thus a non-parametric variable importance scheme for random forests has been proposed in [26, 83] to make it possible to argue about the effects of input variables on the output value. For the permutation accuracy

$$\Delta_{h_2,x}(x,y) = (h_2(x,y) - h_2(0,y))^2 \quad \Delta_{h_2,y}(x,y) = (h_2(x,y) - h_2(x,0))^2$$



$$\int_{-1}^{+1} \int_{-1}^{+1} \Delta_{h_2,x}(x,y) dx dy = 1.27, \quad \int_{-1}^{+1} \int_{-1}^{+1} \Delta_{h_2,y}(x,y) dx dy = 1.75$$

Figure 6.4.: Squared difference functions $\Delta_{h_2,x}(x,y)$ and $\Delta_{h_2,y}(x,y)$.

importance metric the importance of a variable is calculated by permuting the the out-of-bag values of this variable for each tree in the forest. Then the output of the tree is calculated for the permuted dataset and for the original out-of-bag dataset. The importance of the variable is calculated as the percent increase in the accuracy of the model. In [26] this approach is only discussed for classification, however, the approach can also be used for regression. Permuting the variable values removes the information of the variable to break the association of this variable with the output value while leaving the distribution intact [76]. If the estimated values of the model are changed dramatically after permuting the variable values this variable is important in that specific model. This scheme works well for random forests because the averaging effect over many models leads to robust variable importance values.

More recently the permutation variable importance metric for random forests has been studied in more detail using simulation studies with the conclusion that this approach has some undesirable properties [204, 203]. One problem of the original formulation of permutation variable importance is that it assumes that regressors are not correlated. If there are correlations the correlation is broken up because only one of the correlated variables is permuted at each step. In [205, 206] a permutation approach is proposed which groups variables into subsets and permutes all variables of the same subset. With this approach the impact variable subsets can be determined, however, the computational complexity becomes an issue when there is a large number of variable subsets.

6.2.5. Generalized Variable Importance

In [119] it is suggested that the permutation variable importance approach for random forests can be extended to other models and the approach is adopted for artificial neural

networks. Another approach of generalized variable impacts for black box models is described in [172, 207]. In these contributions the idea to calculate variable impacts not only as an average over the whole data set but also for each instance is described. Calculating variable impacts for instances is a powerful concept which can be used to find explanations for class values of specific instances or groups of instances. The method described in [172] is not based on permutations but on direct manipulation of the variable values (e.g. replacement by a representative value). The approaches described in [119, 172, 207] all lead to inaccurate variable impact results when regressors are correlated [209]. Thus in [209] the *Interactions-based Method for Explanation* is introduced which also works for datasets with interactions between input variables. Recently the idea of variable impacts or explanations for specific instances in the dataset has been further pursued [120, 118, 208] in relation to identification of lever variables.

6.2.6. Improved Formulations of Variable Importance Metrics for GP

In this section improved formulations of frequency-based variable relevance Rel_{freq} and impact based variable relevance Rel_{MC} are described. The frequency-based variable relevance is calculated over the whole GP run and has the advantages that it is easy to understand and p-values for variable relevance can be calculated easily. The impact-based variable relevance measure is based on permutation variable importance [26, 119] and is more robust if the model contains conditional expressions or non-linear functions.

Extension of Frequency-based Variable Importance for GP

Given a set of n variables $x_i, i = 1 \dots n$ the frequency-based relevance of a variable x_i is the relative frequency of references to the variable over all models s of the population Pop . Each model can have multiple references to the same variable.

$$\text{freq}(x_i, \text{Pop}) = \sum_{s \in \text{Pop}} \text{CountRef}(x_i, s) \quad (6.1)$$

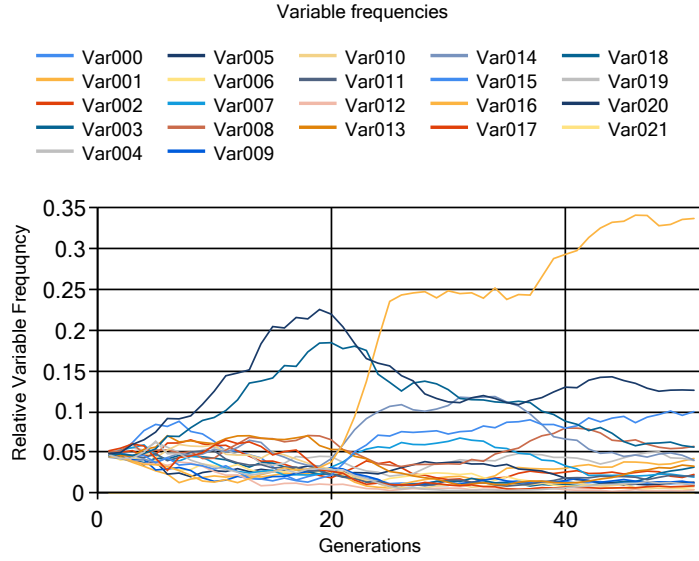
$$\text{rel}_{\text{freq}}^{\%}(x_i, \text{Pop}) = \frac{\text{freq}(x_i, \text{Pop})}{\sum_{k=1}^n \text{freq}(x_k, \text{Pop})} \quad (6.2)$$

The function CountRef returns the number of references to variable x in model s and can be defined recursively for symbolic expressions trees:

$$\text{CountRef}(x, s) = \begin{cases} 1 + \sum_{b \in \text{Subtrees}(s)} \text{CountRef}(x, b) & \text{if Symbol}(s) = x \\ 0 + \sum_{b \in \text{Subtrees}(s)} \text{CountRef}(x, b) & \text{if Symbol}(s) \neq x \end{cases} \quad (6.3)$$

Over the whole GP run the variable relevance can be calculated for each population generation Pop_i and can be traced over time. Tracing the variable relevance over the whole GP run and visualizing the relative frequencies of all variables already can provide a significant insight into the relative importance of the variables and into the dynamics of the GP run itself. Figure 6.5 shows the trajectories of relative variable frequencies

Figure 6.5.: Trajectories of relative variable frequencies over a single GP run for a benchmark dataset.



for an exemplary genetic programming run for a benchmark dataset. The figure shows that at the beginning the relative frequency of two variables rises, however in the later stages the process discovered that another variable, that was not referenced very often in the beginning, is also important to describe the response variable.

When the variable relevance is calculated only for specific generation, for instance only for the last generation of the run, the dynamic behavior of variable impacts is not taken into account. To account for the dynamic effects an average relevance value $\text{rel}_{\text{freq}}(x_i)$ for each variable x_i over m generations can be calculated.

$$\text{Rel}_{\text{freq}}(x_i, \cdot) = \frac{1}{m} \sum_{g=1}^m \text{rel}_{\text{freq}}^{\%}(x_i, \text{Pop}_g) \quad (6.4)$$

Because of the non-deterministic nature of the GP process the relevances of variables over multiple independent GP runs typically differ from each other. Implicit linear or non-linear dependencies between input variables are another possible cause for differences over independent runs. So the variable relevance results of one single GP run cannot be trusted fully. It is desirable to analyze multiple variable relevance results in order to get statistically significant results about which variables are most likely absolutely necessary to explain the response variable, and which variables have a high relevance in single runs only by chance. This can be done through basic statistical analysis over the variable relevance vectors gathered over multiple independent GP runs.

Wilcoxon's signed rank test [230] is used for the statistical analysis of relevances of variable over multiple runs. The goal of the analysis is to find out, if the variable relevance

of each variable found in the GP runs are statistically significant. For this a location test is used to compare the relevance of each variable to the relevance of a reference variable, that is assumed to be irrelevant to model the response variable. The selection of this reference variable is a little bit problematic. A practical approach is to select the variable that has the worst mean relevance over all runs as the reference variable. The relevances are not distributed normally thus the Student's t-test cannot be used in this case. Furthermore, the variable relevances of a single run are not independent from each other and must be treated as a paired value. Wilcoxon's signed rank test can be used for these assumptions.

Impact-based Variable Importance

The impact-based variable relevance metric Rel_{MC} is calculated using permutations of the original values. This variable importance metric is derived from the approach that is followed in random forests (cf. 6.2.4). Using permutation sampling for calculating the variable relevance has the advantage that it works also for non-symmetric and discrete variable distributions and is more robust when the model contains conditionals or non-linear functions. The disadvantage of this method is that the strong assumption, that all variables x_i are independent, must hold.

It should be noted, that impact-based variable reference can be calculated for any kind of estimation model $\hat{y} = g(x)$, even if the structure of g is unknown. Given such a model g the variable impact can always be calculated as only the response \hat{y} of the model for certain x must be available to calculate the impact. This means that the approach is not limited to symbolic regression but can also be used for other regression methods.

Notably, this approach suffers from the same problems as discussed in Section 6.2.4, namely it does not work reliably with dependent input variables as the permutation sampling breaks dependencies in input variables. The issue can be resolved by a more exhaustive permutation sampling approach or using a generalized variable importance approach (cf. 6.2.5).

As suggested in [232] the impacts of all variables can be aggregated over all models of the population, optionally weighting the impact value for a given variable and model with the quality of the model. We propose to calculate the variable impacts only for the final solution. Over multiple independent GP runs the results of the impact-based variable relevance vary. Again we can aggregate the impacts over multiple GP runs and calculate a p-value for the average impact-based relevance using Wilcoxon's signed rank test in the same way as for the frequency-based variable relevance.

6.2.7. Validation of Variable Relevance Metrics

To demonstrate the reliability of the approach we used three different artificial benchmark problems for which the underlying functions are known. First we approximated the influence of all variables on the response value using permutation testing. We executed GP runs with frequency-based variable relevance calculation and impact-based variable relevance calculation for each benchmark problem to see if the relevant variables

Parameter	Value
Population size	1000
Generations	20
Training Samples	2000
Validation Samples	2000
Selection	Offspring selection 50% Proportional 50% Random
Comparison factor	1.0
Success ratio	1.0
Crossover	Sub-tree swapping
Mutation	Single-point
Evaluation	Mean squared error
Evaluation wrapper	Linear scaling

Table 6.1.: Genetic programming parameter settings for the validation of variable relevance metrics.

are identified reliably by genetic programming. We also calculated the $\text{Rel}(\text{impact}_{\text{mean}})$ value as described in [232] for comparison. The goal of the experiments is to find out, if the described variable relevance metrics reliably identify the correct variables, if the order of the relevant variables matches the order of the actual variable impacts and if the relative relevance values are proportional to the actual variable impacts.

Table 6.1 shows the parameter settings for the experiments. For each benchmark problem ten independent runs were executed.

Table 6.2 shows the overall variable relevance results in combination with p-values over ten independent GP runs for the Breiman-I benchmark problem. For comparison the approximated actual variable impacts $\text{Impact}_{\text{fun}}$ for the Breiman-I function are also provided. For the Breiman-I function the signal-to-noise ratio is rather low. Still with $\alpha = 0.01$ the impact-based relevance measures correctly identified five out of seven actually relevant variables. The frequency based relevance metric Rel_{freq} identified only four out of seven variables. The variables x_4, x_7 which have the weakest impact on the response value could not be identified correctly by any relevance measure. The ranking of the relevant variables is correct for all variable relevance measures, however the Rel_{MC} measure is most accurate.

Figure 6.6 shows a kernel density estimation plot generated using the statistical software R. The probability density function for the impact of each variable is estimated from the observed variable impacts for the Breiman-I dataset (the averages are shown in Table 6.2). Kernel density estimates are a way visualize the probability density function based on a limited sample of a random variable. The plot shown in Figure 6.6 also shows that the impacts of variables x_1, x_2 , and x_5 are significantly larger than the impacts of the other variables.

Table 6.3 shows the overall variable relevance results in combination with the p-values

Var	Impact _{fun}	Rel _{mean}	p-Val	Rel _{freq}	p-Val	Rel _{MC}	p-Val
x1	27.333	19.437	0.002	0.286	0.002	24.453	0.002
x2	6.000	3.329	0.002	0.150	0.002	5.293	0.002
x5	6.000	2.591	0.002	0.106	0.004	4.432	0.002
Noise	4.000						
x3	2.666	0.068	0.008	0.082	0.064	0.138	0.008
x6	2.666	0.499	0.008	0.067	0.010	0.971	0.008
x4	0.666	0.000	1.000	0.077	0.064	0.000	1.000
x7	0.666	0.000	1.000	0.060	0.432	0.000	1.000
x8	0.000	0.000	1.000	0.057	0.492	0.000	1.000
x9	0.000	0.000	1.000	0.053	0.695	0.000	1.000
x10	0.000	0.000	1.000	0.051	1.000	0.000	1.000

Table 6.2.: Actual variable impacts Impact_{fun} for the Breiman-I function and variable relevance results over ten independent GP runs.

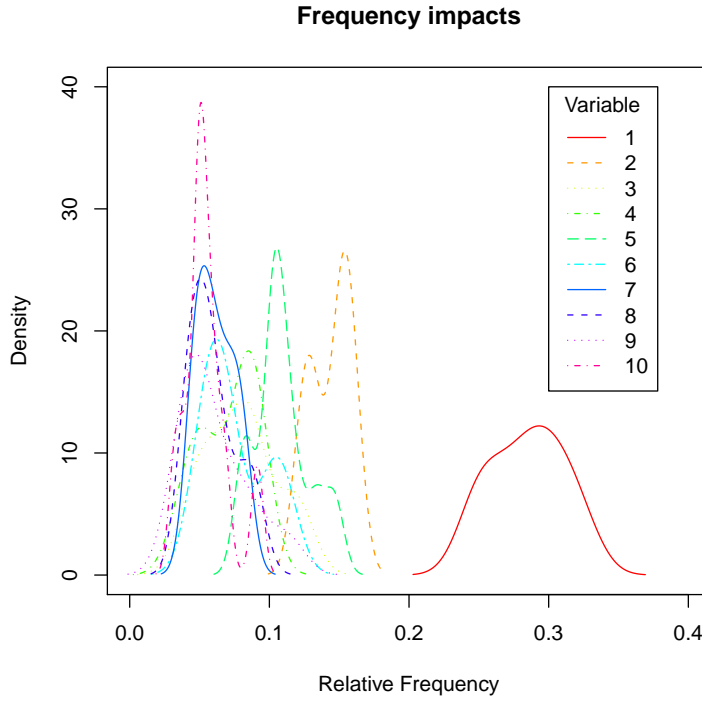


Figure 6.6.: Kernel density estimation of frequency-based variable impacts for the Breiman-I dataset.

Var	Impact _{fun}	Rel _{mean}	p-Val	Rel _{freq}	p-Val	Rel _{MC}	p-Val
x2	6.400	2.956	0.002	0.315	0.002	5.913	0.002
x1	3.860	2.087	0.002	0.211	0.002	3.720	0.002
x3	1.500	0.660	0.002	0.137	0.002	1.319	0.002
Noise	1.000						
x4	0.666	0.260	0.002	0.090	0.002	0.519	0.002
x5	0.166	0.028	0.002	0.049	0.020	0.055	0.002
x6	0.000	0.000	1.000	0.032	1.000	0.000	1.000
x7	0.000	0.000	1.000	0.038	0.432	0.000	1.000
x8	0.000	0.000	1.000	0.041	0.131	0.000	1.000
x9	0.000	0.000	1.000	0.034	0.846	0.000	1.000
x10	0.000	0.000	1.000	0.047	0.002	0.000	1.000

Table 6.3.: Actual variable impacts Impact_{fun} for the Friedman-I function and variable relevance results over ten independent GP runs.

over ten independent GP runs for the Friedman-I benchmark problem. Again the approximated actual variable impacts are also provided. With $\alpha = 0.01$ the impact-based relevance measures identified all actually relevant variables. The variable x_5 with the weakest influence is ranked highly by the variable-frequency based relevance measure, but the result is not significant. Unfortunately with the frequency-based relevance measure the variable x_{10} was also identified as relevant because in the 10 runs the impact value of x_{10} was consistently higher than the impact value of x_6 which was used as the reference variable. Since x_6 and x_{10} are both irrelevant for the problem, this is a false positive of our variable impact routine. The number of false positives should be significantly lower if pruning is used in the GP runs to remove branches with low impacts on the response value.

The variable ranking is identified correctly by all relevance measures. Again the permutation sampling approximation of the variable impact is most accurate.

Table 6.4 shows the overall variable relevance results over ten independent GP runs for the Friedman-II benchmark problem and the approximated actual variable impacts for comparison. The benchmark problem has a high signal-to-noise ratio, so it should be easier for GP to identify the relevant variables. With $\alpha = 0.01$ genetic programming identified and ranked four of the five relevant variables (x_4, x_1, x_2, x_5) correctly regardless of the variable relevance metric. The variable x_3 was not identified correctly, this is the only variable in the Friedman-II function that has a quadratic influence on the function response. The method produced no false positives for this benchmark problem. The relevance metric based on permutation sampling of the variable impact is again most accurate.

Figure 6.7 shows a scatter plot (X-Y plot) of the actual variable impacts and the variable relevance found by genetic programming over all three benchmark problems. The impact based relevance metrics are strongly correlated to the actual impact values. The

Var	Impact _{<i>f_{un}</i>}	Rel _{<i>mean</i>}	p-Val	Rel _{<i>freq</i>}	p-Val	Rel _{<i>MC</i>}	p-Val
x4	16.660	7.401	0.002	0.274	0.002	14.818	0.002
x1	12.970	5.394	0.002	0.177	0.002	10.201	0.002
x2	12.970	4.895	0.002	0.164	0.002	9.156	0.002
x3	4.440	0.000	1.000	0.039	0.557	0.000	1.000
x5	4.160	1.642	0.002	0.133	0.002	3.257	0.002
Noise	1.000						
x6	0.000	0.000	1.000	0.032	1.000	0.000	1.000
x7	0.000	0.000	1.000	0.039	0.557	0.000	1.000
x8	0.000	0.000	1.000	0.037	0.770	0.000	1.000
x9	0.000	0.000	1.000	0.046	0.375	0.000	1.000
x10	0.000	0.000	0.812	0.040	0.322	0.000	0.812

Table 6.4.: Actual variable impacts Impact_{*f_{un}*} for the Friedman-II function and variable relevance results over ten independent GP runs.

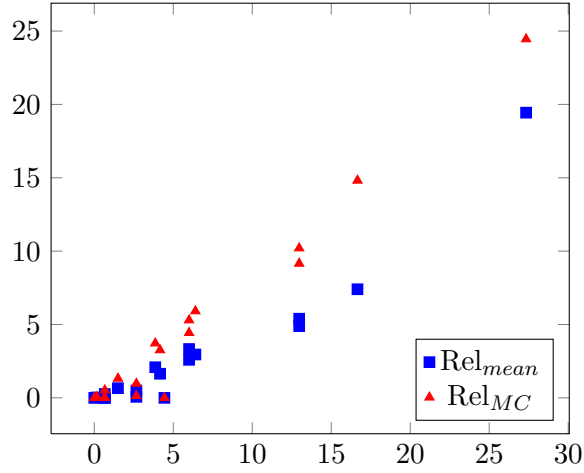


Figure 6.7.: Scatter plot of actual variable impact and variable relevance values. The Rel_{MC} metric is very accurate with a correlation coefficient $R^2 = 0.97$. The Rel_{mean} metric has a correlation coefficient of $R^2 = 0.92$.

Var	Rel _{mean}	p-Val	Rel _{freq}	p-Val	Rel _{MC}	p-Val
x6	15,963.020	0.002	0.154	0.002	13,782.287	0.002
x1	911.585	0.004	0.077	0.006	14,439.618	0.004
x3	68.450	0.016	0.065	0.006	4,736.550	0.016
x4	0.851	0.016	0.053	0.027	6,433.626	0.016
x24	0.000	1.000	0.045	0.233	0.000	1.000
x8	0.000	1.000	0.045	0.014	0.000	0.062
x7	0.000	1.000	0.033	0.106	0.000	0.188
x25	0.000	1.000	0.032	0.106	0.000	1.000
x12	0.000	1.000	0.031	0.037	0.000	0.031
x13	0.000	1.000	0.030	0.106	0.000	1.000
x5	0.000	1.000	0.029	0.233	0.000	1.000
x11	0.000	1.000	0.025	0.322	0.000	1.000
x2	0.000	1.000	0.023	0.233	0.000	1.000
x9	0.000	1.000	0.022	0.275	0.000	1.000
x10	0.000	1.000	0.022	0.233	0.000	0.062
x15	0.000	1.000	0.020	0.770	0.000	1.000
x20	0.000	1.000	0.020	0.557	0.000	1.000
x19	0.000	1.000	0.020	0.084	0.000	1.000
x23	643.499	0.002	0.019	0.131	12,173.905	0.002
x21	0.000	1.000	0.018	1.000	0.000	1.000
x17	0.000	1.000	0.017	0.846	0.000	1.000
x14	0.000	1.000	0.017	0.557	0.000	1.000
x22	0.000	1.000	0.016	1.000	0.000	1.000
x16	0.000	1.000	0.015	1.000	0.000	1.000

Table 6.5.: Variable relevance results for the Dow Chemical tower dataset over ten independent GP runs.

Rel_{MC} is more accurate especially for high impact values. The values of the frequency-based variable relevance metric are not shown in this diagram since they are percentage values.

In this section we described a number of further developed variable relevance metrics based on [232]. We tested the accuracy and reliability of the variable relevance metrics using three artificial regression datasets for which the response generating function is known. The actual variable influence for each benchmark function was approximated using permutation sampling [76] and the resulting value was compared to the values of the enhanced variable relevance metrics. We observed that genetic programming reliably discovered the subsets of relevant variable impacts. Furthermore, the variables were ranked correctly by all relevance metrics, and the impact-based metric based on permutation sampling (Rel_{MC}) is most exact.

6.3. Data Mining and Symbolic Regression

For data mining tasks a target variable is often not known beforehand. Instead the process should identify interesting and relevant relations between variables. Multiple extensions to the simple formulation of symbolic regression have been proposed to allow multiple target variables or to let the GP process identify interesting target variables implicitly.

[101] formulated extensions of symbolic regression for multi-variate targets and demonstrated that GP is able to find solutions to simple multi-variate symbolic regression problems. However the GP literature concentrates on the formulation for a single target variable. Chapter 7 will treat this topic in more detail.

An alternative evaluation scheme which enables GP to search for implicit equations (e.g., $0 = f(x)$) is described in [178]. This approach is remarkable as it is not necessary to specify target variables, instead the process will find functional expressions which evaluate to a constant. This approach will be described in more detail in the following section.

The most trivial way to approach a data mining task is to try to find independent symbolic regression models for all variables. Models that accurately match the original variable values are potentially more interesting and thus should be presented to the data miner. However, the large number of unconnected regression models will most likely lead to confusion. Additionally, if there are correlated variables this process will just output a model, where the single correlated variable is used to explain the target variable. This behavior is most likely unwanted since such relations are most likely already known. Instead the process should be able to recognize such trivial relations and search for models that are more interesting.

Instead of independent genetic programming runs, resulting in an unconnected set of symbolic regression problems for all variables, it is more desirable to combine the results in order to allow a data miner to gain a full understanding of the interrelations in the dataset. In Section 6.5 a cooperative process for GP-based data mining is described.

6.4. GP-Based Search for Implicit Equations

Genetic programming has been used to search for implicit equations in the form of $0 = f(x, y)$ [178, 180]. This is essentially a way to search accurate models for multiple target variables. For example if a function $f(x, y) = 0$ is to be found, the target variables are x and y . To make sure that the process does not converge to trivial functional expressions (e.g., $1 - 1$, $2 * x - x - x$) partial differentials of the model for all target variables are evaluated and compared to the numeric approximation of the differential. The error measure is defined as the sum of errors over all partial differentials [84, 179]. The drawback of this approach is, that differentials of variables must be approximated numerically. Especially in the presence of noisy data the noise will be further amplified by trivial numeric differential approximation methods based on finite differences. If the data contains noise, the problem can be alleviated through smoothing methods (e.g. using

the locally weighted scatter-plot smoother [38, 39]) or more sophisticated interpolation methods. However, it is likely, that relevant information in the data is lost through smoothing or interpolation methods. This method has been applied successfully to find equations for well known physical systems (e.g., pendulum, spring pendulum) [180].

6.5. Comprehensive Search for Symbolic Regression Models

Given a dataset that contains observed values from a system, the usual objective is to approximate one target variable using a subset of the possible input variables. This is often problematic because of implicit relations in the set of input variables. Pairwise correlated variables can be detected easily but the input variables might be related in a non-linear fashion which is difficult to detect. Implicit relations in the input variables have the effect that it is not possible to produce one unique ideal model for the target variables. Instead it is possible to produce equally accurate and correct models using alternative representations resulting from the implicit relations.

A simple but very powerful way to approach this issue is to search for all accurate models over the whole dataset. Instead of a single target variable treat each variable as the target variable and try to create an approximation model using all other variables as potential inputs. With this approach it is possible to uncover non-linear implicit relations which might can be used interchangeably in models for the target variables. This can also lead to a better understanding of the overall system because hierarchical structures in approximation models are made explicit and can be studied in detail.

6.5.1. Case Study: Chemical-II Dataset

In this section the unguided approach to generate symbolic regression models is demonstrated on the Chemical-II dataset. This dataset has been prepared and published by Arthur Kordon, research leader at the Dow Chemical company. The dataset contains 4999 observations of 26 variables of a chemical process and can be downloaded from <http://vanillamodeling.com/realproblems.html>. The following description of the Tower dataset also stems from the same source.

The observations in the dataset stem from a chemical process and include temperatures, flows, and pressures. The target variable is the propylene concentration and is a gas chromatography measurement at the top of a distillation tower. The propylene concentration is measured in regular intervals of 15 minutes. The actual sampling rate of the input variables is one minute, but 15 minutes averages of the inputs are used to synchronize with the measurements of the target variable. The range of the measured propylene concentration is very broad and covers most of the expected operating conditions in the distillation tower.

Modeling

A screening of the dataset for correlated variables shows that the dataset contains four groups of variables with strong pairwise correlations. For each group one variable is se-

Variable cluster
x₂₀ , x_{21}
x₁₂ , x_{13} , x_{16}
x₃ , x_7 , x_{11}
x₈ , x_{25}

Table 6.6.: Groups of variables with strong pairwise correlations in the Chemical-II dataset.

lected as the representative and the remaining variables are not considered for modeling. Table 6.6 lists the identified clusters of strongly correlated variables.

For modeling the full dataset is partitioned into a training partition (rows 100–2050) used to calculate model fitness, an internal validation partition (rows 2050–4000) used to select the final model, and a test partition (rows 4000–4999) used to estimate the generalization error of the final model.

Table 6.7 shows GP parameter settings for the experiments. The final model (best on validation) is linearly scaled to match the location and scale of the original target values. This is necessary because the squared correlation coefficient, which is invariant to scale, and location is used as fitness function. As shown by [95] maximization of the squared Pearson’s Product Moment Correlation Coefficient is equivalent to minimization of the scaled mean squared error. Static length and depth constraints are used for the models to prevent unlimited code growth. The initialization procedure PTC2 [128] creates random trees with a target distribution for the tree length between 3 and 100 nodes.

The same parameter settings are used to create models for each remaining variable of the dataset. Ten independent GP runs are executed for each configuration, leading to 190 symbolic regression models, produced in the same number of GP runs.

Results

Figure 6.8 shows a box-plot of the R^2 of the models produced by GP for each of the variables over 10 independent runs. From the box-plot it is immediately clear that the dataset contains a number of implicit variable relations. GP consistently identified almost perfect models for variables $x_3, x_5, x_6, x_8, x_9, x_{12}$ and x_{20} . GP also identified a number of highly accurate models with squared correlation coefficient larger than 0.9 for variables x_1, x_4 , and x_{22} . Models produced by GP for the response variable have on average $R^2 = 0.87$. For variables x_{19}, x_{15} , and x_{24} GP also consistently produced relatively accurate models. For the remaining variables the R^2 of the GP models varies strongly, especially for variable x_2 no accurate models are found.

The box-plot can give an indication about the ability to approximate variables when other variable values are known. The full power of the unguided approach is more apparent when the information gathered in all GP runs is combined to produce the variable relation network shown in Figure 6.9. In the network the most relevant input variables to approximate a variable are shown where $a \rightarrow b$ means a is a relevant variable for GP models approximating variable b. The network is based on the permutation impact of

Parameter	Value
Population size	2000
Max. generations	100
Parent selection	Tournament
	Group size = 5
Replacement	1-Elitism
Initialization	PTC2
Crossover	Sub-tree-swapping
Mutation rate	15%
Mutation operator	One-point
	One-point constant shaker
	Sub-tree replacement
Tree constraints	static limits
	Max. depth = 10
	Max. length = 100
Model selection	Best on validation
Fitness function	R^2 (maximization)
Function set	+, -, *, /, avg, log, exp
Terminal set	constants, variables

Table 6.7.: Genetic programming parameters for the Chemical-II dataset.

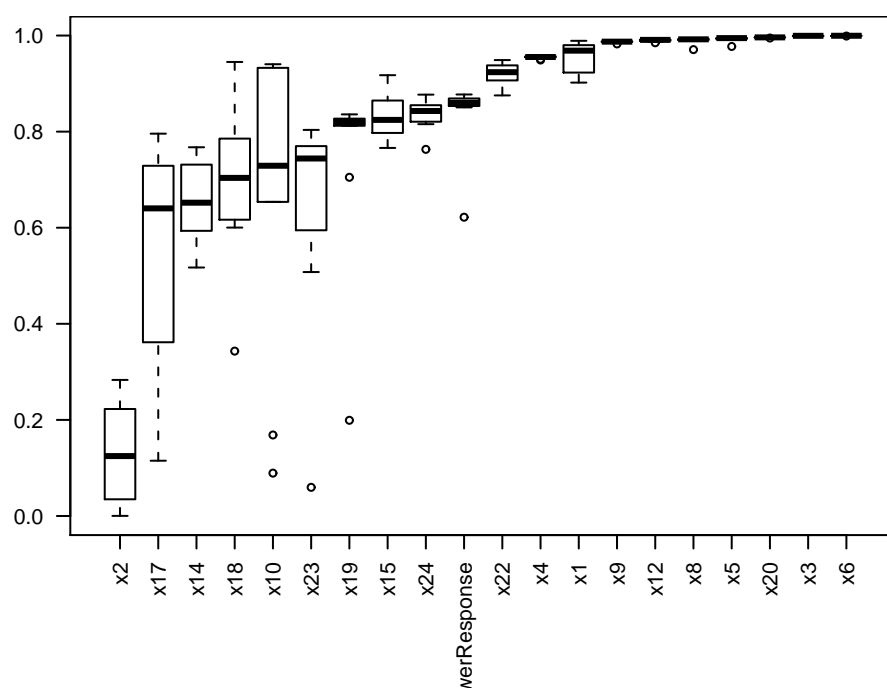


Figure 6.8.: Box-plot of R^2 on the test set of the model output and original values for the Chemical-II dataset.

variables in the final model. For a given variable x the permutation impact of all other variables is calculated over the 10 models for x . Only significant impacts (p-Value < 0.005) are kept and shown in the network diagram. The variable clusters identified in the preliminary analysis of the dataset are also shown as rectangles containing all variables of the cluster. In depth analysis of the relationships identified by GP showed, that variables x_1 and x_6 and the two clusters with representatives x_8 and x_3 are all pairwise strongly connected. So a larger cluster has been manually introduced that encompasses all those variables.

Analyzing the network generated for the Chemical-II dataset in more detail, leads to a better overall insight into the process. An immediate observation is, that the large cluster with representative x_1 and the cluster with representative x_{12} play a central role in the process. Additional variables with many outgoing arrows are x_{10} and x_{19} . These two variables are often used to approximate other variables.

Another fact that is immediately clear is, that there are many double-linked variable pairs and two long double-linked chains of variables. This indicates that the two variables are strongly related. One chain relates the cluster with representative x_{20} with x_5 which is in turn strongly connected to x_{14} . Another chain connects variables x_{17} , x_{22} and x_{24} . Other strongly related variable pairs are x_{18} and x_{19} , and x_{10} and x_{23} .

The relevant input variables for models for the response variable are x_{23} and one variable in the large central cluster with variable x_1 .

The network diagram shows the general relations of variables in the dataset. For more detailed information the models produced for each variable should be analyzed in detail.

Result Details

In this section a number of manually selected and simplified highly accurate models produced for the Chemical-II dataset are presented in detail.

Table 6.8 gives an overview of the presented models including the squared correlation coefficient (R^2) on the test set.

GP identified simple but highly accurate linear models for variables x_6 (6.10), x_{12} (6.13), and x_{20} (6.14). GP also identified a linear model with $R^2 = 0.87$ for the response variable using only six input variables (6.16). For comparison we also trained a ν -SVM regression model [222] using the same six input variables with five-fold cross-validation and a grid search for the parameters ν , C , γ using the open source SVM implementation libSVM [32]. The best SVM model we found has a R^2 of 0.97 which is significantly better than the linear model. Motivated by the SVM result we also trained a more complex GP model using only the subset of six variables. The GP runs produced a non-linear model (6.17) which after some manual simplification has a $R^2 = 0.90$.

It is now possible to deconstruct the model for the response variable further because accurate models for all input variables are available. The original model uses variables x_1 , x_6 and x_{12} for which accurate approximations are available. It is possible to replace the input variables by their approximation models and so get an alternative approximation model for the response. Through this procedure the model does not necessarily become more complex because, if the introduced models contain common terms these

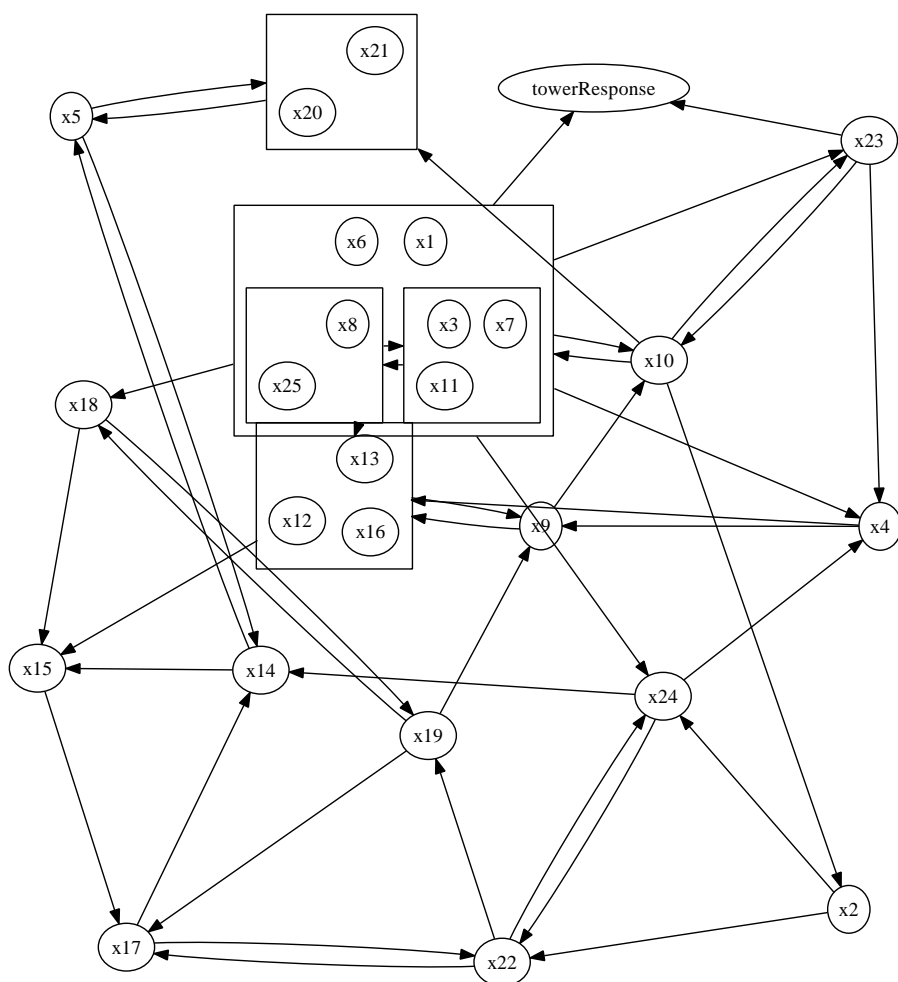


Figure 6.9.: Identified variable relations for the Chemical-II dataset.

Target	Model	R^2 (test)
x_1	(6.5)	0.97
x_3	(6.6)	0.99
x_4	(6.8)	0.91
x_5	(6.9)	0.99
x_6	(6.10)	0.96
x_8	(6.11)	0.99
x_9	(6.12)	0.98
x_{12}	(6.13)	0.98
x_{20}	(6.14)	0.99
x_{22}	(6.15)	0.94
Response	(6.16)	0.87
Response	(6.17)	0.90

Table 6.8.: Accuracy of selected and simplified models for the Chemical-II dataset on the test set.

can be unified. In this case all approximations contain the variable x_3 which does not occur in the original model for the response variable. This hierarchical approach of model analysis leads to a more throughout understanding of the possible impact factors and implicit relations, that are relevant for the approximation of the response variable.

$$x_1 = \frac{x_6}{(x_3 + x_8 + x_{10})\left(\frac{x_8}{x_{10}} + \frac{x_3}{(x_3+x_{10})(x_3+x_4+x_6)}\right)} \quad (6.5)$$

$$(6.6)$$

$$x_3 = \frac{\log(x_8)}{x_1 + x_6 + x_8 + x_{10}} \quad (6.7)$$

$$x_4 = \frac{x_3 + x_{24}}{x_{23}^2} \quad (6.8)$$

$$x_5 = \log(x_3 + x_9 + x_{10} + x_{20}) \quad (6.9)$$

$$x_6 = x_3 + x_8 \quad (6.10)$$

$$x_8 = \frac{1}{x_1 + x_3 + x_6 + x_{10} + x_{24} + \frac{1}{x_3+x_6}} \quad (6.11)$$

$$x_9 = x_3 + x_{12} + x_{15} + x_{19} \quad (6.12)$$

$$x_{12} = x_3 + x_9 + x_{20} \quad (6.13)$$

$$x_{20} = x_3 + x_5 + x_{10} + x_{12} \quad (6.14)$$

$$x_{22} = \frac{1}{\frac{x_{17}}{x_1+x_{19}+x_{10}+c_0} + \frac{x_9+\exp(x_{17}+x_4+x_{24}+c_1)+c_2}{x_{17}^2}} \quad (6.15)$$

$$\text{Response} = x_1 + x_2 + x_6 + x_{12} + x_{15} + x_{23} \quad (6.16)$$

$$\begin{aligned} \text{Response} = & x_2 + x_6 + x_{15} + \frac{1}{x_2} + \frac{1}{x_{12}} + \frac{x_1^3 + x_1^2}{x_1 + x_6} \\ & + \frac{x_1 x_2^2}{x_1 + x_2 + x_{23}} + \frac{x_{23} x_6}{x_1 x_{12}} \end{aligned} \quad (6.17)$$

6.6. Improving GP Performance

The training time of data-based modeling algorithms is generally considered a non-issue. In the whole process of knowledge discovery from databases the major part of the time is spent in data preparation and interpretation and analysis of models, only a small part of the time is spent in training [70]. Even though the time it takes for training is not a large concern there is still a benefit in improving the training performance as long as the quality of the final model is the same. Especially for the task of finding all interesting and relevant models without a predetermined target variable a large number of GP runs have to be executed. For this task reducing the training time means that more models for all variables can be generated in the same time and the chance to find potentially interesting models is higher.

In general two approaches are possible to increase the performance of GP, parallelization and reduction the computational effort of fitness evaluation. Fitness evaluation is usually the most expensive step in GP and the performance of the algorithm is bound by the performance of the fitness function.

One especially effective approach which has been pursued intensively recently and

combines both aspects is genetic programming on GPUs [235, 236, 113, 80, 37, 115]. The recent developments in hardware and software have made powerful massively parallel architectures accessible for general purposes and facilitated such research.

6.6.1. Parallelization

Evolutionary algorithms in general are embarrassingly easy to parallelize because of the concept of independent solutions which are combined to a population. A number of parallel and distributed execution schemes have been described for evolutionary algorithms in general [7, 139] which can also be applied to genetic programming [13, 151, 160, 91, 201, 46, 34].

6.6.2. Improving Fitness Evaluation Performance

Often a number of preliminary experiments with a reduced training set are carried out to find good parameter values for a algorithm before the final model is trained using the full training set.

It is also possible to reduce the number of fitness cases dynamically in a single GP run. The fitness value of an individual is usually only an approximation of the actual fitness of the individual even when all fitness cases are considered. Thus, if an accurate approximation is also possible when only a small number of fitness cases are evaluated the performance of the algorithm can be increased.

Reducing the number of fitness cases has already been discussed by Holland who introduced a method for the “optimal allocation of trials” in [86] which is based on a solution of the multi-armed bandit problem. The method tries to find a good approximation for the fitness with a limited amount of fitness cases. Later an extended approach called “rational allocation of trials” has been introduced in [212]. In [75] it has been shown that the number of fitness cases that is needed in the GP fitness function can be upper-bounded via statistical and information-theoretic considerations. In [98] it has been shown that a very simple scheme that randomly selects a small subset of fitness cases (e.g. 10% of the training partition) for each evaluation also work well for specific applications and has a large effect on performance.

Recently an approach for the reduction of fitness cases specifically for symbolic regression has been introduced [223]. The aim of this approach is not primarily to improve the performance of the algorithm but to produce more robust models by intelligent data balancing. In the case of symbolic regression a data-set often contains a large number of similar data points and the outer regions of the data-space are populated only sparsely. Data balancing weights the fitness cases by relevance metrics based on the distance of a data-point to the remaining data points and removes redundant data-points completely in order to put more emphasis on the outer regions of the data-space. Effectively the number of fitness cases can be removed and the performance of fitness evaluation is increased.

Another approach to increase fitness evaluation performance is to use fitness predictors which can for instance be co-evolved with the actual solution candidates [181].

In conclusion it is not necessary to use all fitness cases in symbolic regression and a number of different approaches for the reduction of fitness cases have been described in the literature. In our experiments we found that the scheme of randomly sampling a small subset of fitness cases for each fitness evaluation is simple and effective and does not have a negative effect on final solution quality.

7. Multi-Variate Symbolic Regression and Time Series Prognosis

Prediction is very difficult, especially about the future.

Niels Bohr

Multi-variate symbolic regression is an extension to genetic programming which increases the size of the hypothesis space. The difference is that multiple variables can be selected as target instead of only one in the case of simple symbolic regression. Multi-variate symbolic regression models are encoded in the same way as normal symbolic regression models, as symbolic expression trees. The difference is that the result producing branch has multiple subtrees. The scalar results of each branch are combined to an output vector. The n -th sub-tree of the result producing branch is a simple symbolic regression model for the n -th target variable. The shape of a multi-variate symbolic regression solution is shown in Figure 7.1. Figure 7.2 shows a multi-variate symbolic regression solution that also includes automatically defined functions (ADF) [105] which can be called in all component branches of the result producing branch (RPB).

The multi-variate variant of symbolic regression has been described already in [101]. It has been shown that simple problems (e.g. approximating the result of the vector product function) can be solved with this extended form of symbolic regression [101]. In this chapter we discuss the benefits of multi-variate symbolic regression in more detail and describe how a real-world problem can be solved efficiently with this formulation.

If the target variables are independent, there is no benefit in using the multi-variate approach as it is easier to create separate scalar models in independent GP runs for each

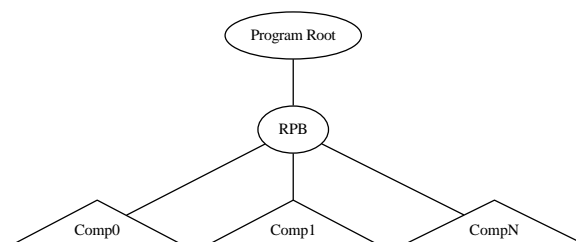


Figure 7.1.: A multi-variate symbolic regression model has a separate branch for each component of the result vector below the result producing branch (RPB).

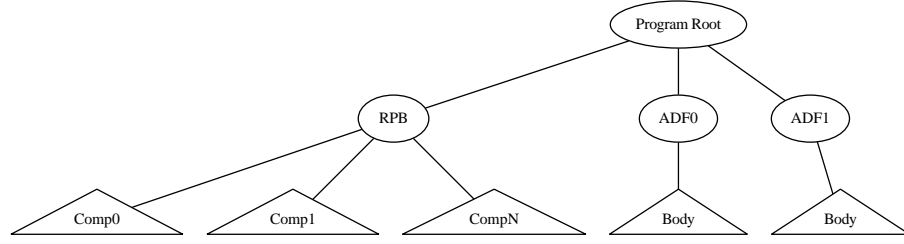


Figure 7.2.: Multi-variate symbolic regression model with result producing branch (RPB) and two automatically defined functions (ADF0, ADF1)

target variable. If the variables are independent it is futile to try to find common building blocks which can be used in models for the different variables. In such cases nothing can be gained from a combined formulation where a combined model for multiple target variables is evolved in a single GP run.

If there are dependencies between the target variables, the multi-variate formulation might be beneficial. This is common in technical systems where output values are often influenced by the internal state of the system and, as such, can not be treated as independent measurements. If one is interested in both output variables then two separate models for both output variables have to be created, using simple symbolic regression. With multi-variate symbolic regression a single GP run produces a combined model for both output variables. In cases like this, the benefit of the multi-variate approach is that the GP process can evolve code fragments which can be used to estimate multiple target variables. The effect is, that the single GP run to create a combined model is more efficient than multiple independent runs. Additionally, if automatically defined functions [101] are allowed, the reuse of code fragments can be made more explicit through common ADFs which can be referenced by all component-models. Through code reuse in multiple models, combined with the power of ADFs to extract common code fragments and make them explicit in form of ADFs, the possibility arises to create multi-variate hierarchical models in which the system hierarchy is mapped to the model hierarchy. The aim of multi-variate symbolic regression in the context of knowledge discovery is thus not solely to make the process more efficient, but to generate models in which common building blocks are explicit as such hierarchical models are easier to interpret. Common code fragments in ADFs reduce the amount of code that has to be analyzed and understood, in order to understand the combined model for all target variables.

7.1. Evaluation of Multi-Variate Symbolic Regression Models

Multi-variate symbolic regression can be formulated either as single-objective or multi-objective optimization problem. The multi-objective formulation is straight-forward. The estimation error is calculated for each component independently and a multi-

objective algorithm is used for minimization of component errors. The result is a Pareto-front of non-dominated solutions with minimal errors in components. The multi-objective approach directly works correctly for correlated variable pairs and for differently scaled variable values. With the multi-objective approach it should also intuitively be easier to find hierarchical models and explicit code fragments in ADFs. One algorithm that can be used for multi-objective optimization of such models is NSGA-II [48].

For the single-objective formulation an aggregation function for the errors of the components is necessary, in order to calculate a scalar quality value which represents the overall quality of the model. A simple solution is to sum up the mean squared errors over all components. The MSD is defined as the mean of the squared Euclidean distances over all rows of the target values and the estimated values. The sum of mean squared errors over all m components is thus the same as the mean squared Euclidean distance over all rows shown in equation 7.1.

$$\begin{aligned} \text{MSD} &= \frac{1}{n} \sum_{i=1}^n L_2(y'_i, y_i)^2 \\ L_2(x, y) &= \sqrt{\sum_{d=1}^m (x_d - y_x)^2} \end{aligned} \tag{7.1}$$

The Euclidean distance cannot be used if the target variables have different scales or locations. In such cases the component errors are weighted unfairly, so the prediction error of the target variable with the larger scale would dominate the overall quality value. The component errors have to be weighted to make sure that each component error has the same impact on the overall solution quality, regardless of the scaling and location of the original values. One solution is to scale all variable values to the same range in a preprocessing step. This is often recommended for various data-analysis methods to improve the solution quality. In the case of genetic programming and multi-variate symbolic regression the scaling can also be integrated into the process. The mean and variance of the original values are calculated for each target variable and used to scale the original and the estimated values to a random variable with zero mean and standard deviation of one. The aggregation of the mean squared errors of the scaled values is the normalized mean squared error for multi-variate regression models (see Section A.1.1).

The NMSE for multi-variate regression models has drawbacks when a sub-set of the target variables is strongly correlated. An example should help to make this easier to understand. Obviously the output of a model for three target variables should match the original values as closely as possible. Two of the variables are strongly correlated while the third variable is uncorrelated. If the model matches only one of the correlated variables well, it can be argued that the model does not accurately match the analyzed system as the implicit dependency between the two correlated target variables is not modelled correctly. The NMSE treats all components as independent thus a quality improvement of any component leads to a proportional quality improvement of the whole model. The Mahalanobis distance (7.2) also accounts for correlations between variables

[130]. It can be used as a quality measure for multi-variate models in place of the NMSE. To calculate the Mahalanobis distance between an original row vector and the estimated row vector a covariance matrix for the original target variables is necessary. The covariance matrix C is calculated from the original vales. If the target variables are independent, the covariance matrix is diagonal, so the Mahalanobis distance boils down to the NMSE. If the target variables are all normalized to $N(0,1)$, the covariance matrix is the identity matrix. The mean Mahalanobis distance of the estimated values of a model to the original values is larger, if the estimated values only match one of the correlated target variables closely as the component distances are weighted by the covariance.

$$L_M(x, y) = \sqrt{(x - y)^T C^{-1} (x - y)} \quad (7.2)$$

7.1.1. Curse of Dimensionality

The multi-variate normalized mean squared error and Mahalanobis distances can be used up to a limited number of dimensions. For high-dimensional problems the quality metrics are problematic because of the distribution of distances in high dimensional spaces [5, 223]. An improved fractional distance metric (L_f norm) for high-dimensional spaces (7.3) has been proposed in [5], which can be used for multi-variate modeling.

$$\text{dist}_{f,d}(x, y) = \left(\sum_{i=1}^d (x_i - y_i)^f \right)^{\frac{1}{f}}, f \in (0, 1) \quad (7.3)$$

7.2. Multi-variate Time Series Modeling and Prognosis

The multivariate regression approach is especially suited for the prognosis of multivariate time series. The aim of time series prognosis is to predict the values of a covariate x for future time points $(x_{t+1}, x_{t+2}, \dots, x_{t+h})$ up to a certain horizon h based on previous values (x_0, x_1, \dots, x_t) . Time series prognosis models in most cases include an auto-regressive term. This means, the model includes previous values of x for the prediction of future values. Equation (7.4) shows a simple linear auto-regressive model for x_{t+1} using n previous values $(x_t, x_{t-1}, \dots, x_{t-n+1})$ with model parameters a_i weighting the relative contributions of the previous values. Note that a given AR model (with fixed parameters a_i) can be used to predict future values of x up to any horizon $x(t+h)$ by simple recurrence using the values $(\hat{x}_{t+1}, x_t, x_{t-1}, \dots, x_{t-n+2})$ to calculate \hat{x}_{t+2} and so on.

$$\hat{x}_{t+1} = \sum_{i=0}^{n-1} a_i x_{t-i} + \epsilon_t \quad (7.4)$$

Genetic programming can be used to create symbolic time series models. Depending on the type of the underlying system from which the time series was measured and

the intended application of the model, different modeling approaches are possible. The approach discussed here, is useful for the identification or approximation of dynamic systems. This task is closely related to symbolic regression, however, because of the system dynamics the previous values of x can have an influence on y as well, so the input values of the model have to be extended to include previous values of x up to a maximal lag l . Equation (7.5) shows a model for a dynamic system, in comparison to regression models the time parameter is relevant and previous values of x are included as model inputs. In order to create dynamic models with genetic programming, the terminal set has to be extended to include terminals for lagged variables up a certain maximal lag limit. For dynamic models the same fitness functions, which are used for symbolic regression, can be used (e.g. MSE).

$$\hat{y}_t = f(x_t, x_{t-1}, \dots, x_{t-l}) + \epsilon_t \quad (7.5)$$

If the dynamic system includes a feedback loop, the response y_t of the system not only depends on the values x_t and previous values of x_{t-i} but also on previous output values y_{t-i} . So the model has to be extended to also include previous values of y up to a maximal lag m and the result is the auto-regressive model 7.6.

$$\hat{y}_t = f(x_t, x_{t-1}, \dots, x_{t-l}, y_{t-1}, \dots, y_{t-m}) + \epsilon_t \quad (7.6)$$

To create auto-regressive models with genetic programming the terminal set has to be extended to also include terminal symbols for lagged values of y . This kind of model can be used to predict the future values of y when the future values of the input / control variables are known. The question that the model can answer is: What is the response of the system for future values of the control variables x and current and previous values of x and y . The estimation \hat{y}_{t+1} can be feed back as input into the model and in combination with the inputs x_{t+1} the estimated response \hat{y}_{t+2} at time $t+2$ is produced.

The model cannot be used to predict future values of y when future values of x are not known yet. In some applications it is sufficient to predict y only up to a limited horizon h . In such cases a compromise is possible by using values of x only up to time step x_{t-h} as input (7.7). With this model it is possible to predict the next h values of y based on the previous values of x and y . The disadvantage of this approach is that recent values of x cannot be included in the approximation, the larger the prediction horizon the larger the gap between input-values and estimated values.

$$\hat{y}_{t+h} = f(x_t, x_{t-1}, \dots, x_{t-l}, y_{t+h-1}, \dots, y_{t-m}) + \epsilon_t \quad (7.7)$$

For this reason it is sometimes preferable to create fully auto-regressive models without exogenous variables 7.8. This model has no explicit target variable, all values are used as input and produced as output. Similarly to multi-variate regression, we can easily create such multi-variate time series models with genetic programming. Note that for a fixed model it is possible to predict future values of x to any finite horizon through recurrence (even though the accuracy of the prognosis is expected to deteriorate with increased time steps $(t+i)$ because of the accumulation of approximation errors ϵ_{t+i}).

$$\hat{x}_{t+1} = f(x_t, x_{t-1}, \dots, x_{t-l}) \quad (7.8)$$

7.2.1. Evaluation of Multi-variate Time Series Prognosis Models

In genetic programming fitness evaluation of auto-regressive models is often problematic because of the implicitly high correlation of y_t and y_{t-1} in time series datasets. In symbolic regression the fitness of a solution is usually calculated as an average distance over n pairs of original and approximated value for a data-set with n rows (e.g. MSE). In time series modeling this fitness function assigns a relatively high fitness value to pathological solutions, where the output value \hat{y}_t is only dominated by the input value y_{t-1} because of the high correlation of two subsequent values of y . The result is that such solutions (super-individuals) dominate the population and the algorithm quickly converges. The final solution is practically useless because the system dynamics are not modelled correctly and the solution cannot be used to calculate a meaningful prognosis.

The core of the problem is that it is easy to produce an accurate one-step prediction, using the simple random walk model ($\hat{y}_t = y_{t-1} + \epsilon_t$). To find better models, which represent the system dynamics, it is necessary to evaluate time series models in an environment where the random walk model is not a super-individual. In fact, the random walk model should be used as a base line result. It is the worst possible model that is accepted only when the time series cannot be predicted consistently (the original time series is a random walk). In conclusion the random walk model is not better than any other randomly initialized model of the first generation. So the fitness of the random walk should be similar to the fitness of a randomly initialized model. We assume here, that the original time series does not have a linear trend, however, the argument also holds for time series with a long term trend.

One approach to evaluate time series models which is motivated by this idea, is to calculate the fitness of a model on the basis of on longer sequences of predicted values. For each point of the training set a series of predicted values up to horizon h is calculated instead of a single one-step prognosis. The idea is that if the underlying dynamics of the time series are identifiable, then it should be possible to find a model that consistently outperforms the random walk model for the larger prediction horizon h .

Linear Scaling of Time Series Prognosis Models

Linear scaling is a fitness evaluation wrapper originally formulated for symbolic regression [95]. The raw output values of the model are linearly scaled to match the location and scale of the original target values using the two parameters α and β (intercept and slope) (7.9). The scaled mean squared error of the output values and the original target values is used as fitness function, in order to make it easier for the evolutionary process to find models that match the shape of the target variables. If the MSE of the unscaled output values is used as fitness function the evolutionary process first concentrates to create models that produce output in the correct range of the target values, reducing genetic diversity in the progress. Only after the initial stages, the evolutionary process

starts to match the shape of the target values. This issue is solved with linear scaling as the output values are transformed to the correct location. Minimization of the scaled mean squared error is equivalent to maximization of Pearson's product moment correlation coefficient [95]. The correlation coefficient can be calculated in a single pass over the predicted and estimated values while the scaled mean squared error needs two passes. One pass is necessary to calculate the parameter values α and β , and a second pass is necessary to calculate the scaled mean squared error.

$$\begin{aligned} \text{SMSE}(y, \hat{y}) &= \frac{1}{n} \sum_{i=1}^n (y_i - (\alpha + \beta \hat{y}_i))^2 \\ \beta &= \frac{\text{Cov}(y, \hat{y})}{\text{Var}(\hat{y})} \\ \alpha &= \bar{y} - \beta \bar{\hat{y}} \end{aligned} \tag{7.9}$$

Linear scaling can also be applied to time series prognosis. This is not straight forward if the model has auto-regressive terms, because in this case the estimated output value at point t is used as input for the prediction of the output value at point $t + 1$ and the scaled estimated values must be used as input instead of the raw estimated value. This issue can be solved by calculating the scaled error in two steps. The algorithm for linear scaling of time series models is proposed for the first time in Algorithm 7. First the raw one-step predictions are calculated for the whole training set. Then α and β are calculated based on the one-step predictions and the original target values. The resulting scaling parameters are used to calculate the full prognosis up to the prediction horizon for each point in the second pass over the dataset, using the scaled predicted value to override the original measured values in y . The set of tuples $(w(h), y[t], y[t+h], v[t+h])$ containing the original value at the start of the prognosis $y[t]$, the original value at the current horizon $y[t+h]$, the scaled predicted value at the current horizon $v[t+h]$, and an optional weighting $w(h)$ is used to calculate the fitness of the model.

One straight forward fitness function, that can be used in this algorithm, is the mean squared error over all predicted values 7.10. Please see A.1.2 for more advanced fitness functions that can be used to calculate the fitness after the model output values have been calculated.

$$\text{MSE}(P) = \frac{1}{|P|} \sum_{(w(h), y[t], y[t+h], v[t+h]) \in P} (y[t+h] - v[t+h])^2 \tag{7.10}$$

7.2.2. Case Study: Financial Time Series Prognosis

In this section we demonstrate the application of GP-based multi-variate time series prognosis on a financial time series dataset. The dataset has been prepared by Ricardo de A. Araújo and Glaucio G. de M. Melo for a financial time series prediction competition, which was held as a side event of the EvoStar conference 2010 [43]. Only little information

Algorithm 7: Linear scaling and fitness evaluation of time series prognosis models

input : Model: m , Fitness function: f , Column vector: y_n , Matrix:

$X = [x_{i,j}]_{n \times k}$, Maximal lag: l_{max} , Horizon: h_{max}

output: Fitness of m

```
// subset of  $y_n$  for which the model can be evaluated
target  $\leftarrow y_{l_{max} \dots n}$ ;
// calculate one step predictions
 $\hat{y} \leftarrow (m(y_{i-l_{max} \dots i-1}, X_{i-l_{max} \dots i \times k}) | i \leftarrow l_{max} \dots n)$ ;
// calculate linear scaling parameters
 $\beta \leftarrow \frac{\text{Cov}(\text{target}, \hat{y})}{\text{Var}(\hat{y})}$ ;
 $\alpha \leftarrow \overline{\text{target}} - \beta \overline{\hat{y}}$ ;
 $P \leftarrow \emptyset$ ;
for  $t \leftarrow l_{max}$  to  $n - h_{max}$  do
     $v_n \leftarrow y_n$ ;
    for  $h \leftarrow 1$  to  $h_{max}$  do
         $v[t+h] \leftarrow \alpha + \beta m(v_{t+h-l_{max} \dots t+h-1}, X_{t+h-l_{max} \dots t+h \times k})$ ;
         $P \leftarrow P \cup (w(h), y[t], y[t+h], v[t+h])$ ;
    end
return  $f(P)$ ;
end
```

about the origin of the dataset has been published, except that the dataset stems from real financial time series from the Brazilian stock exchange.

The original dataset contains 200 observations of ten financial time series. The original competition objective was to predict the next 10 data-points for each of the 10 time series. Unfortunately the 10 data points, which were originally held back by the organizers to evaluate the entries, have never been published. So we repartitioned the dataset into a training partition (rows 1–190) and a test partition (rows 190–200).

Modeling

Table 7.1 shows the GP parameter settings for the time series prognosis experiments. In this experiment static size and depth limits are used to prevent unlimited code growth. The size limit is rather large (1000 nodes) because a single multi-variate model has ten branches, one for each component of the multi-variate time series. The random models of the initial population are created with the probabilistic tree creator with a uniform target distribution of tree sizes between 1 and 1000 nodes. The function set contains arithmetic operators including a special function for the calculation of the arithmetic mean and additionally logarithm, exponential and sine function symbols. The terminal set includes random constants, lagged variables and derivatives with allowed time offsets between (t-27) and (t-1) and additionally terminal symbols to calculate the moving-average and integral of the values of a variable over a given time span within the range

Parameter	Value
Population size	10000
Max. generations	100
Parent selection	Tournament Group size = 5
Replacement	1-Elitism
Initialization	PTC2
Crossover	Sub-tree-swapping
Mutation rate	15%
Mutation operator	One-point One-point constant shaker Sub-tree replacement
Tree constraints	Static limits Max. depth = 10 Max. size = 1000
Model selection	Best on validation
Fitness function	scaled NMSE Prediction horizon = 10
Function set	+, -, *, /, avg, log, exp, sin
Terminal set	const, lagged variable, ma, integral, derivative Time offsets: (t-27)–(t-1)

Table 7.1.: Genetic programming parameter settings for the financial time series prognosis experiments.

of allowed time offsets. For instance the expression $MA(x_1, -10, -5)$ is the moving-average of the variable x_1 in the time span from (t-10) up to (t-5). The derivative values of a variable are approximated numerically from the original values without smoothing (see A.2). The model is auto-recursive because all target variables are allowed also as input variables, and the output values of the model for time t are used as input values to calculate the output for time $t + 1$. The maximal prediction horizon is ten time steps. Models which do not reuse output values can be created if the time lags in all terminals are smaller than -10.

For the GP runs the training partition spanning 190 rows is further partitioned into a dataset that is used for fitness evaluation (rows 30–130) and an internal validation set (rows 130–180). In total only 150 rows are used for training, because 30 rows have to be skipped in the beginning to allow for dynamic models including time offsets, and 10 rows have to be trimmed at the end because the prediction horizon is 10. Figure 7.3 shows the model of evaluation for the 10-step prognosis of the financial time series. At each point t of the training partition a prognosis consisting of 10 predicted values for $(t + 1) \dots (t + 10)$ is calculated.

The fitness function is the scaled normalized mean squared error for 10-step predictions summed over the ten time series. Each component-branch of the model is linearly scaled to fit the location and scale of that component of the original multi-variate time series as described in algorithm (7). Scaling of branch output is based on the one-step predictions of the model. After scaling the 10-step prognosis of the resulting model for each of the

	30	31	32	33	34	35	36	...	129	130
y	28.6	28.0	27.5	26.7	26.5			...	21.1	20.8
$y'(t = 30)$	26.8	27.0	26.8	26.7	26.6					
$y'(t = 31)$		27.0	27.0	26.9	26.8	26.7				
$y'(t = 32)$			27.0	27.0	26.9	26.8	26.7			
\vdots								\ddots		
$y'(t = 129)$									20.5	20.5
$y'(t = 130)$										20.5

Figure 7.3.: Model of evaluation for the 5-step prognosis of financial time series.

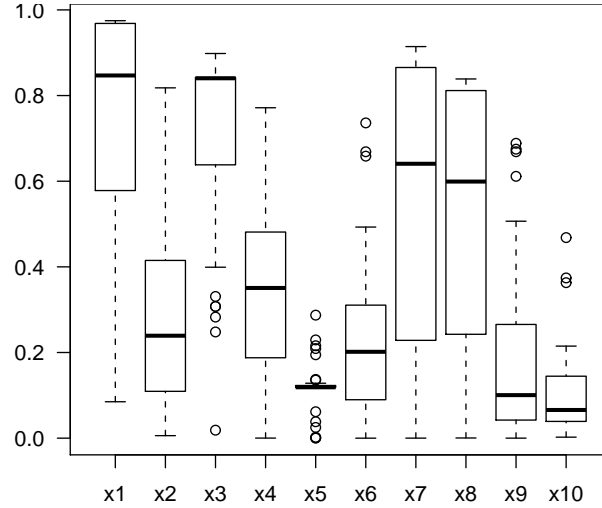


Figure 7.4.: Box-plot of squared correlation coefficient (R^2) over 80 models for each component of the multi-variate financial time series.

training rows (30–100) is calculated (70 (rows) \times 10 (steps) predictions). Finally the normalized mean squared error of the 700 predicted values and the actually observed values is calculated for all ten components of the multi-variate time series and summed up to produce the scalar fitness value. Also see sections A.1.1 and A.1.2 in the appendix for further information on the fitness functions for multi-variate time series prognosis.

Results

Figure 7.4 shows a box plot of the squared correlation coefficient (R^2) of the model output and original values over 80 models produced by GP for each component of the multi-variate time series. The box-plot shows that for some components the output of the multi-variate models has a large squared correlation coefficient (x1, x3, x7, x8), while for other components it is relatively low (x2, x5, x10).

It is tempting to conclude from the rather high R^2 values for some components, that the model can be used to accurately predict future values for these components. However,

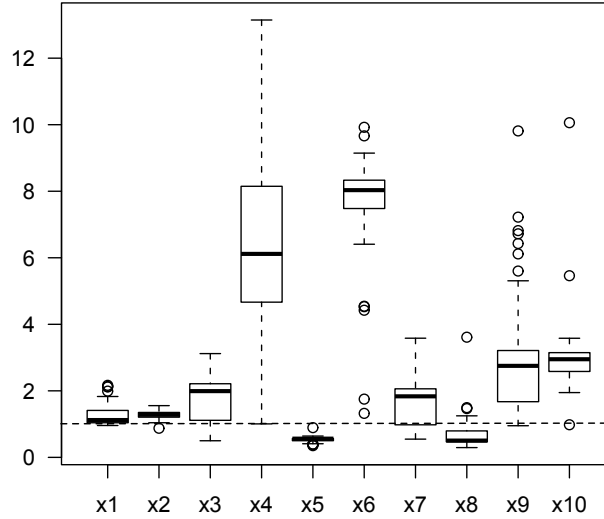


Figure 7.5.: Boxplot of Theil's U over 80 models for each component of the multi-variate financial time series. The horizontal line indicates the Theil's U of the naive prognosis.

the squared correlation coefficient cannot give a good insight about the actual accuracy of the prognosis produced by the model. An accuracy metric that is better suited to measure the accuracy of the predicted values is Theil's inequality coefficient [213] (cf. Section A.1.2). This metric relates the prognosis produced by the model to a naive prognosis. We choose to compare the output of our models with the no-change model. The interpretation of Theil's U is very simple. A perfect prognosis has $U = 0$, the naive prognosis has $U = 1$, and anything that is worse than the naive prognosis has $U > 1$. The results for the multi-variate time series prognosis models produced by the GP experiments are shown in the boxplot of the Theil's U statistic for each component in figure 7.5. This boxplot gives a more realistic indication of the accuracy of the models when used for prognosis of the financial time series. It is immediately clear, that the multi-variate models produced by the GP runs are actually not very useful for 10-step prognosis of the financial time series. The models are consistently worse than the naive no-change prediction for more than half of the variables (x3, x4, x6, x7, x9, x10). For two variables the multi-variate models generated by GP produce a prognosis that is not better than the naive prognosis. Only for two components (x5 and x8) the prognosis of the GP models is consistently better than the naive prognosis.

Detailed Results

In the following the model with the lowest SMSE on the validation set is presented in more detail. Equation (7.11) shows the simplified expressions for each component of the time series. For better interpretability constant factors and offsets have been removed. The model is very simple, x3, x8, x9 are approximated using only a linear trend (the

variable Date is the row index). Variables x5, x6, x7 and x10 are approximated using either the integral symbol or the moving average symbol for smoothing.

Figure 7.6 shows line charts of the original variable values and the 10-step prediction of the model (7.11) for each component over the whole dataset. The model matches the long term trend of the original values relatively well but fails to predict the short term variations except for variable x7.

Figure 7.7 shows line charts of the actual continuation of the time series in the test set (rows 190–200) and the continuation predicted by the model at time $t = 190$. This line chart shows that the model is not capable to predict the short-term future values of the multi-variate time series very well because the model only approximates the long term trend. For most variables the predictions of the model are far off the actual values and the model produces almost constant predictions for most variables. For variables x3, x4, and x7 the model predicts the trend in the wrong direction. The prediction has a positive trend, even though the variable values are actually decreasing.

In conclusion, the time series prognosis approach where GP is used to produce multi-variate models succeeded in predicting the long term trend of the multi-variate time series correctly, however, the short term variations are not predicted correctly.

$$\begin{aligned}
x_1(t) &= x_{10}(t - 10) + \text{Date} \\
x_2(t) &= x_3(t - 24) \\
x_3(t) &= \text{Date} \\
x_4(t) &= x_9(t - 8) \times x_8(t - 20) \\
x_5(t) &= x_7(t - 1) + I(x_6, -14, -4) + \text{Date} \\
x_6(t) &= I(x_6, -26, -4) + MA(x_8, -24, -7) \\
x_7(t) &= \frac{1}{I(x_7, -8, -1)} \\
x_8(t) &= \text{Date} \\
x_9(t) &= \text{Date} \\
x_{10}(t) &= I(x_7, -8, -1)
\end{aligned} \tag{7.11}$$

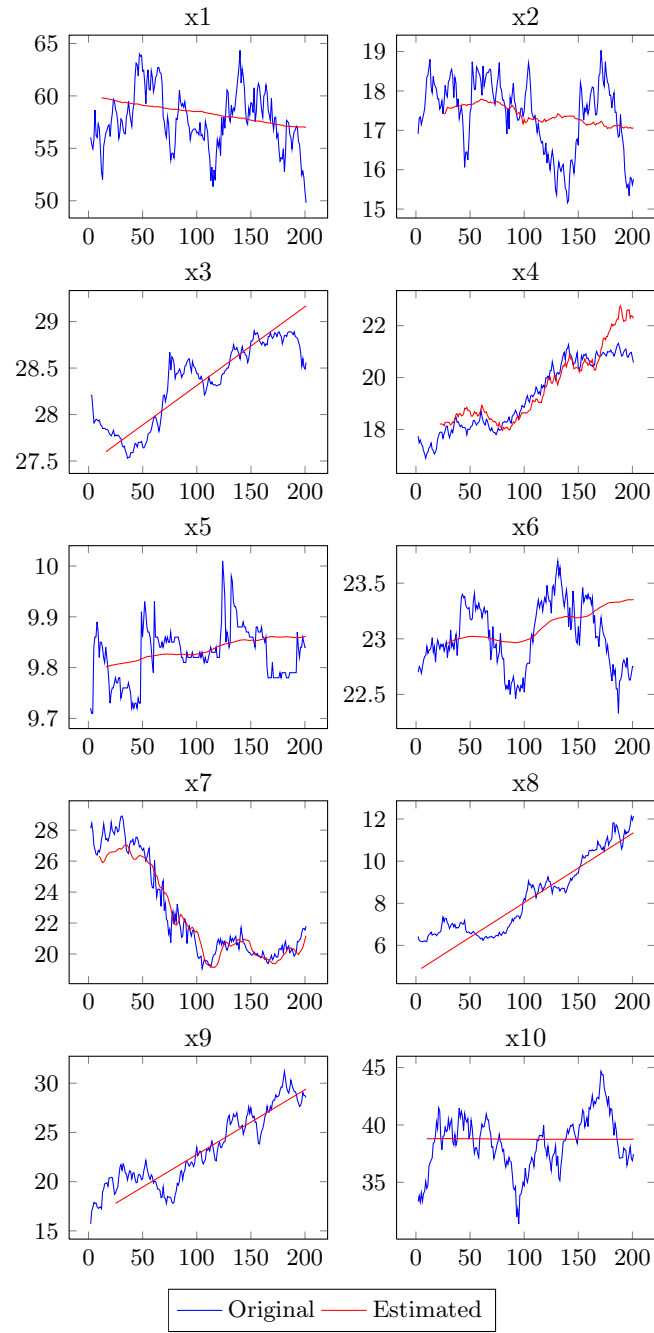


Figure 7.6.: Original values and predicted values (horizon=10) of the model (7.11) for the financial time series dataset.

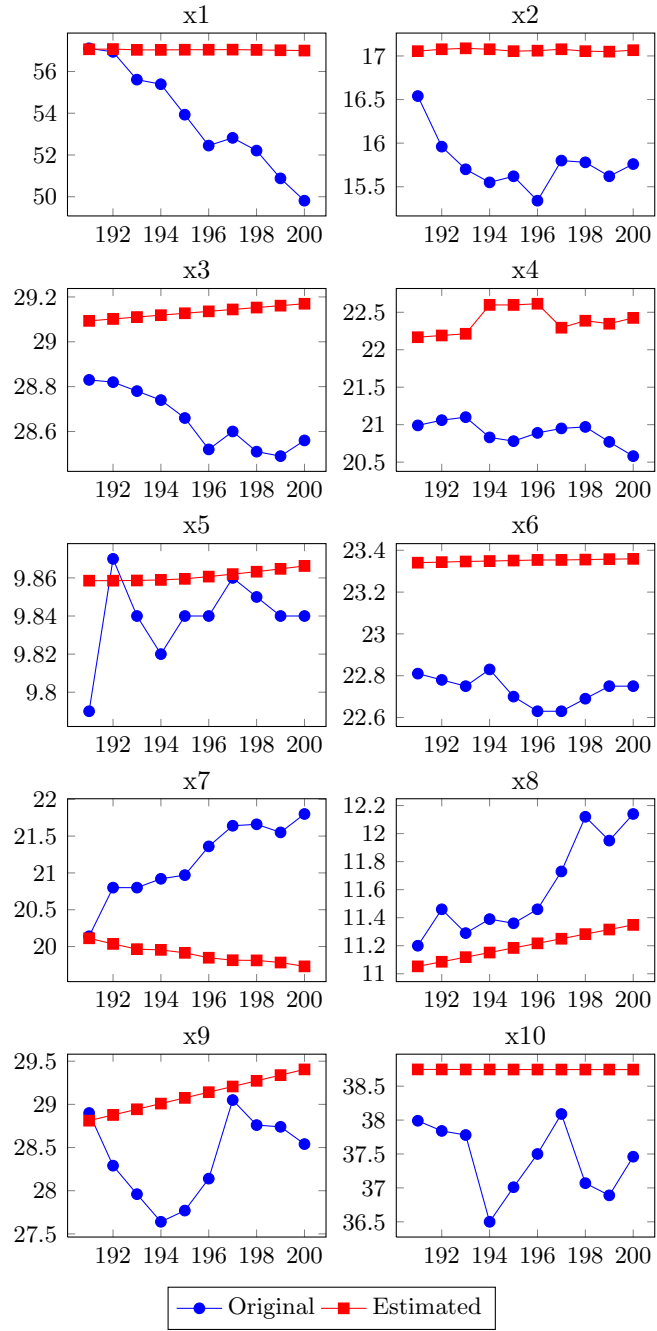


Figure 7.7.: Actual continuation and predicted continuation (horizon=1..10) of the model (7.11) for the test partition (rows 190–200) of the financial time series dataset.

8. Applications and Case Studies

The best thing about being a
statistician is that you get to play in
everyone's backyard.

John Tukey

Genetic programming with the enhancements discussed in the preceding chapters can be used effectively to identify interrelations between variables of complex systems. In this chapter we apply the methods discussed in the previous chapters to analyze: the blast furnace process for iron production, an industrial chemical process, and interactions of macro-economic variables.

8.1. Blast Furnace

Parts of this section have been published by the author in [106]. The author thanks Christoph Feilmayr for his expert input about the blast furnace process and for critically reviewing draft versions of this section.

The blast furnace is the most common process to produce hot metal globally. Around 881 million tons of hot metal and 1240 million tons of steel were produced in 2006 [199]. More than 60% of the iron used for steel production is produced in the blast furnace process [182].

The raw materials for the production of hot metal enter the blast furnace via two paths. At the top of the blast furnace ferrous oxides and coke are charged in alternating layers. The ferrous oxides include sinter, pellets and lump ore. Additionally feedstock to adjust the basicity is also charged at the top of the blast furnace. In the lower area of the blast furnace the hot blast (air, 1200 °C) and reducing agents are injected through tuyeres. Reducing agents include heavy oil, pulverized coal, coke oven or natural gas, and coke tar. Sometimes other reducing agents, like waste plastics, are added to substitute coke. The products of the blast furnace are liquid iron (hot metal) and the liquid byproduct slag, tapped at the bottom, and blast furnace gas which is collected at the top.

Figure 8.1 shows a schematic diagram of a blast furnace for the production of pig iron (image taken from [106]).

Inside the blast furnace the raw materials undergo a number of physical and chemical reactions. The ferrous oxides and coke gradually move down into the lower and hotter parts of the furnace. In this process the mechanical properties of ferrous oxides change because of pressure, temperature and the reducing gas atmosphere. The oxygen contained in ferrous oxides is removed by the reducing agents carbon, carbon monoxide and

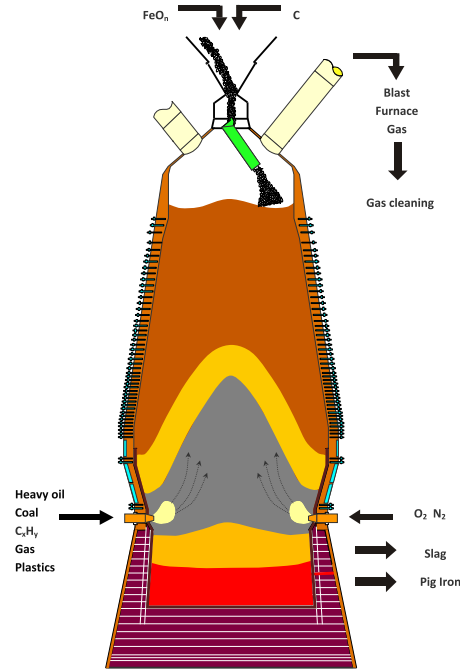


Figure 8.1.: Schematic diagram of the blast furnace and input and output materials. At the top ferrous oxides and coke are charged in alternating layers. Near the bottom the hot blast and reducing agents are injected through the tuyeres. Hot metal and slag are tapped in regular intervals from the hearth of the blast furnace. Blast furnace gas is collected at the top of the blast furnace.

in minor proportion hydrogen. Carbon monoxide is created through partial oxidation of reducing agents and from carbon dioxide and carbon (Boudouard-reaction). In the upper part of the blast furnace the carbon monoxide reduces the ferrous oxides creating carbon dioxide. The resulting metallic iron is melted and carburized. For more details about the process see [202, 155, 184].

The general process and the physical and chemical reactions occurring in the blast furnace are quite well understood. On a detailed level many of the inter-relationships of different parameters and the occurrence of fluctuations and unsteady behavior in the blast furnace are not totally understood. An example is the change in the cooling losses via the walls of the furnace. Changes in the cooling losses are strongly related to the efficiency of the process. However the causes for the effect are not totally understood. Knowledge about such effects can be used to improve various aspects of the blast furnace process, e.g. the quality of the products, the amount of resources consumed, or the stability of the process.

A number of mathematical and simulation models have been developed for specific processes in the blast furnace [11, 12, 192]. A survey of recent progress on mathematical models for the blast furnace is given in [219]. Additionally data-based methods have been applied to model certain aspects of the blast furnace process. The data-based approaches

include neural networks [170, 35, 225, 123, 177], genetic algorithms [158, 176], support vector machines [226], and genetic programming [106].

8.1.1. Modeling

In this section we describe the application of unguided genetic programming to identify interesting interrelations in the blast furnace process, using the principles discussed in the previous chapters.

The basis of our analysis is a dataset containing hourly measurements of the set of variables of the blast furnace listed in Table 8.2. The dataset contains almost 5500 rows, rows 100–3800 are used for training and rows 3800–5400 for testing. Only the first half of the training set (rows 100–1949) is used to determine the accuracy of a model. The other half of the training set (rows 1950–3800) is used for validation and to select the final model. The dataset is not shuffled because the observations are measured over time and the nature of the process is implicitly dynamic.

In each GP configuration one of the variables listed in Table 8.2 is treated as the target variable and all remaining variables are allowed as input variables. This leads to 23 different configurations, one for each target variable. For each configuration 30 independent runs have been executed. Table 8.1 lists the parameter settings for the GP algorithm. Dynamic depth limits are used to increase model parsimony and the correlation between training and validation fitness is used as overfitting indicator. If the correlation decreases to a value below 0.2, the run is stopped before reaching the maximum number of generations (150). The model with the largest R^2 on the validation set is linearly scaled to fit the location and scale of the target variables and returned as the result of the GP run.

8.1.2. Results

Figure 8.2 shows a box-plot of the model accuracy (R^2) over 30 independent runs for each target variable of the blast furnace dataset. The R^2 values are calculated from the response of the validation-best model of each run on the test set. Whiskers indicate four times the inter-quartile range, values outside of that range are indicated by small circles in the box-plot.

Almost all models for the hot blast pressure result in a perfect approximation ($R^2 = 1.0$). Very good approximations are also possible for the O_2 propotion of the hot blast and for the flame temperature. The hot blast temperature, the coke reactivity index, and the amount of water injected through tuyeres cannot be modelled accurately with this approach.

Figure 8.3 shows the network of the most relevant input variables for each target variable. Where an arrow $a \rightarrow b$ indicates that variable a is a relevant variable for approximating variable b . For each target variable the three most relevant input variables are shown. The variable relevance is determined using the frequency-based relevance measure described in Chapter 6.

Parameter	Value
Population size	2000
Max. generations	150
Parent selection	Tournament Group size = 7
Replacement	1-Elitism
Initialization	PTC2
Crossover	Sub-tree-swapping
Mutation rate	15%
Mutation operator	One-point One-point constant shaker Sub-tree replacement
Tree constraints	Dynamic depth limit Initial limit = 7
Model selection	Best on validation
Stopping criterion	$\text{Corr}(\text{Fitness}_{\text{train}}, \text{Fitness}_{\text{val}}) < 0.2$
Fitness function	R^2 (maximization)
Function set	$+, -, *, /, \text{avg}, \text{log}, \text{exp}$
Terminal set	constants, variable

Table 8.1.: Genetic programming parameters for the blast furnace dataset.

Group	Variable
Hot blast	pressure
	O_2 proportion
	speed
	amount
	temperature
Tuyere Injection	total humidity
	amount of heavy oil
	amount of coal tar
	amount of water
Charging	coke charge weight
	amount of pellets
	amount of lump ore
	amount of sinter
	amount of coke
	coke reactivity index
Tapping	burden basicity B2
	hot metal temperature
	amount of slag
Blast furnace top gas	amount of alkali
	temperature
Process parameters	gas utilization CO
	melting rate
	cooling losses (staves)

Table 8.2.: Variables included in the blast furnace dataset.

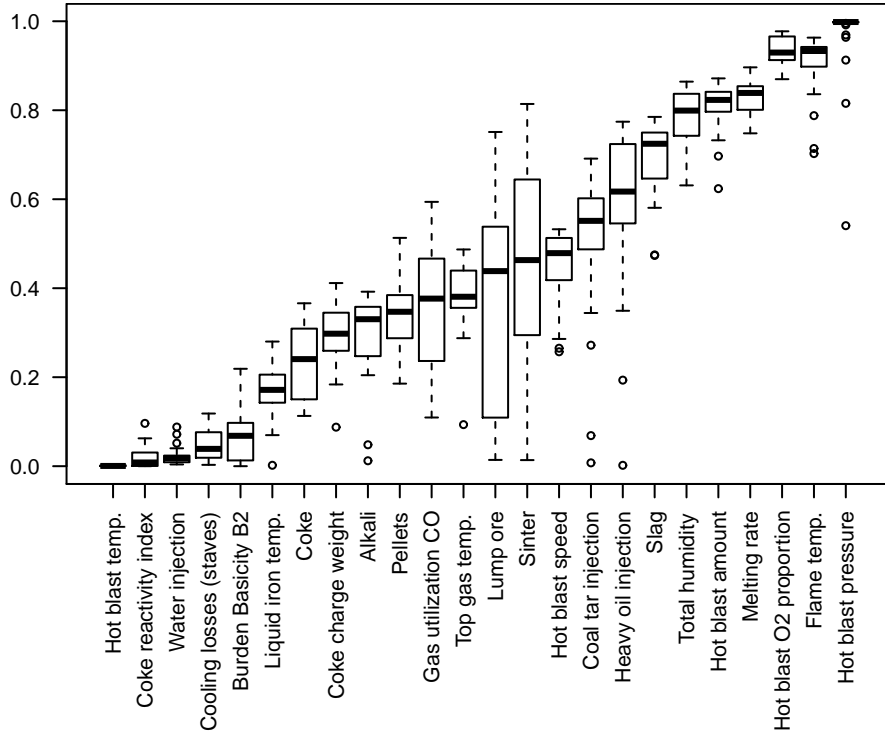


Figure 8.2.: Box-plot of R^2 value on the test set of models for the blast furnace dataset.

Some variables in the network are connected by arrows in both directions. This is an indicator that the pair of variables is strongly related. The value of the first variable can be approximated, if the value of the second variable is known and vice versa.

Variables that have many outgoing arrows play a central role in the process and can be used to approximate many other variables. In the blast furnace network central variables are the melting rate, the amount of slag, the amount of injected heavy oil, the amount of pellets, and the hot blast speed and its O_2 proportion.

In Figure 8.3 all target variables are displayed to present the full results of the experiments. In a data mining scenario it is a good idea to filter the results dynamically, showing only results for target variables for which accurate models are available. The unfiltered variable relationship network should be interpreted in combination with the box plot in Figure 8.2, because the significance (not in the statistical sense) of arrows pointing to variables which cannot be approximated accurately, is low. One example are the cooling losses of the staves. The three most relevant input variables for models for the cooling losses are: the gas utilization (CO), the amount of alkali, and the amount of heavy oil injected through tuyeres. This result has to be viewed critically because all models for the cooling losses, which have been found in the process, are rather inaccurate (R^2 value of approximately 0.2). In any case, the knowledge gained by this experiment can be used to study the effects of these three variables on the cooling losses in more detail.

8.1.3. Result Details

The relationship network for the blast furnace process presented in Figure 8.3 provides a good overview of the blast furnace process. However, many interesting details are not shown in the network because this would add visual clutter to the figure and make it hard to interpret the results. For a more thorough analysis it is necessary to look at the models in more detail. In this section we present a number of selected models for a subset of target variables for which accurate models have been identified. The models presented in this section are simplified versions of the original models produced by GP. The simplification includes manual pruning of branches with minor impact on the prediction and automatic simplification. Table 8.3 gives an overview of the models presented below.

It is important to note that the models presented in Equations 8.1 to 8.12 are the direct result of a data-analysis process and can only show relations of variable values in the dataset in a statistical sense. In particular, the models do not necessarily approximate actual physical or chemical relations or causations in the blast furnace process.

The models show a strong relation of the melting rate with the hot blast parameters. The melting rate is used in models for the hot blast parameters: pressure (8.1), O_2 -proportion (8.5), the hot blast amount (8.9), and the total humidity (8.8) which is largely determined by the hot blast. In return the hot blast parameters play an important role in the model for the melting rate (8.11). Parameters of the hot blast are directly controlled by the operator who adjusts the amount, the temperature, and the O_2 proportion of the hot blast [62]. As such the models identified for the hot blast are not interesting for

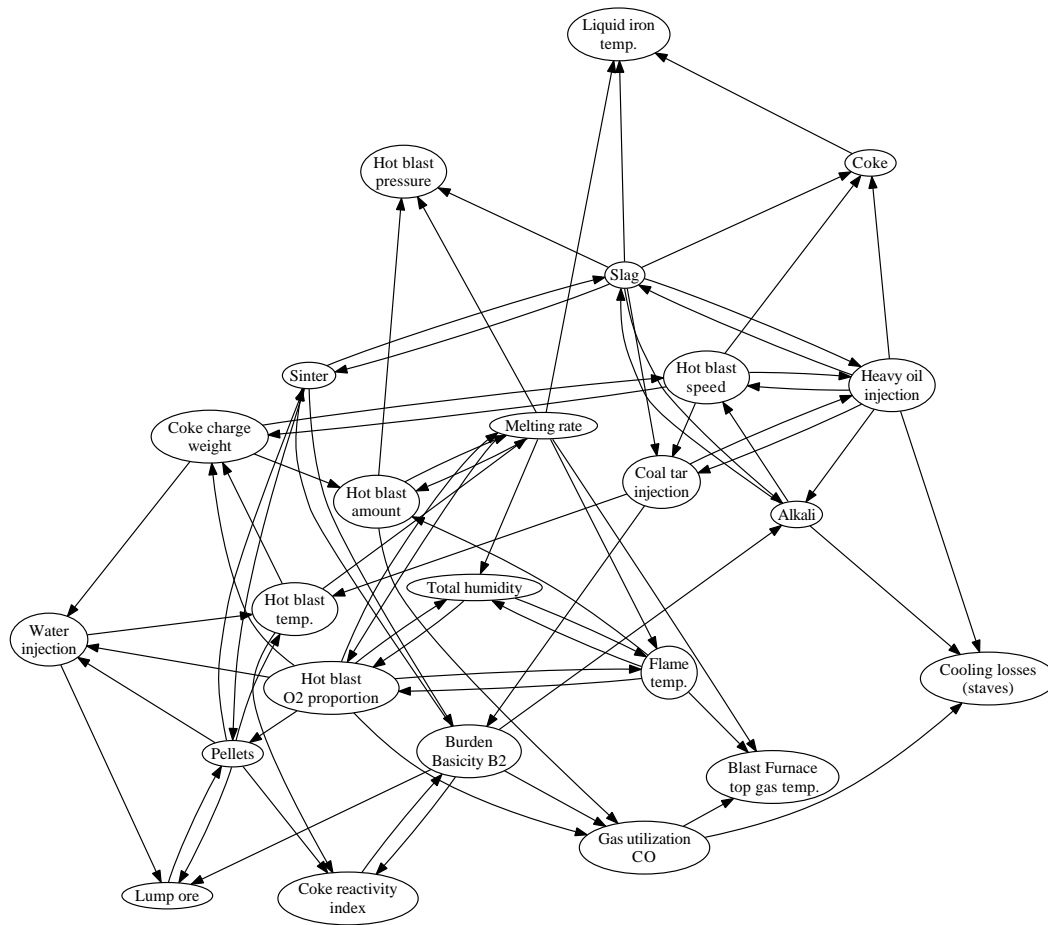


Figure 8.3.: Relationships of blast furnace variables identified with unguided GP-based data mining. Arrows indicate a dependency in the aspect of data modeling and do not necessarily match physical or chemical causations.

Variable	Model	R^2
Hot blast pressure	(8.1)	1.0
Hot blast O_2 -proportion	(8.5)	0.98
Flame temperature	(8.6)	0.96
Total humidity	(8.8)	0.86
Hot blast amount	(8.9)	0.87
Melting rate	(8.11)	0.89
Amount of slag	(8.12)	0.77

Table 8.3.: Overview of selected models for the blast furnace process.

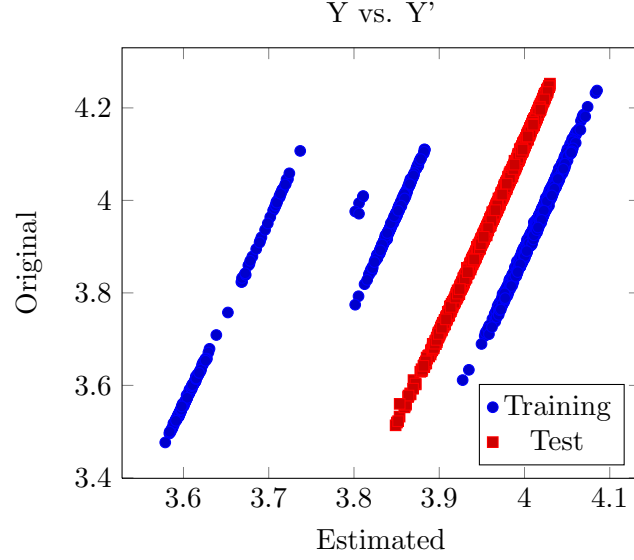


Figure 8.4.: Scatter plot of the original values of $\text{Pressure}_{\text{HB}}$ and the output of model 8.1.

the blast furnace operator [62]. However, they are helpful for the interpretation of other models as they represent the implicit relations of different variables which might be used interchangeably by GP in models for more interesting variables like the melting rate.

The pressure of the hot blast is a result of the hot blast amount and temperature and permeability of the burden through which the hot blast is pressed [62]. The model identified using symbolic regression is shown in Equation 8.1. It has a very high correlation with the original measured values considering only the test partition.

$$\begin{aligned} \text{Pressure}_{\text{HB}} &= c_0 \log \frac{c_1 \text{Speed}_{\text{HB}}}{\text{Amount}_{\text{HB}} \times \text{Temp}_{\text{HB}}} + c_2 \\ c_0 &= -11.4233 \\ c_1 &= 0.79701 \\ c_2 &= -68.9197 \end{aligned} \tag{8.1}$$

The scatter plot of the original values and the output of this model is shown in Figure 8.4. As can be seen in the scatter plot the value is approximated correctly for the test set, however, an additional unknown term which is not included in the model also influences the pressure of the hot blast. This missing term is the tuyere area which changes intermittently every two to six months and stays constant over large time spans [62]. As can be seen in Figure 8.4 the tuyere area is changed five times over the whole period from which the blast furnace data set was collected.

Model 8.1 can be further simplified to the expression shown in Equation 8.2.

$$\text{Pressure}_{\text{HB}} = \alpha(-\log(\text{Speed}_{\text{HB}}) + \log(\text{Amount}_{\text{HB}}) + \log(\text{Temp}_{\text{HB}})) + \beta \tag{8.2}$$

Rows	α	β
1–788	11.35	-67.57
788–1175	11.00	-64.09
1175–1182	9.70	-56.39
1182–1473	10.68	-62.68
1473–5450	11.42	-67.80

Table 8.4.: Scaling parameters α and β for the model for $\text{Pressure}_{\text{HB}}$ shown in Equation 8.2 for different partitions of the blast furnace data set.

Using the full data set we determined the parameters α and β in order to fit the scale and location of the model output for the differences caused by changes in tuyere area. The resulting parameter values for α and β are shown in Table 8.4.

The models shown Equations 8.1 and 8.2 approximate the pressure of the hot blast based on the hot blast amount, speed, and temperature. However, the model disregards the known causal relationship of the three variables shown in Equations 8.4 and 8.3.

$$\text{Amount}_{\text{HB}}^{\text{actual}} = \text{Amount}_{\text{HB}}^{\text{norm}} \times \frac{\text{Temp}_{\text{HB}} + 273}{273} \times \frac{1.013}{\text{Pressure}_{\text{HB}} + 1} \quad (8.3)$$

$$\text{Speed}_{\text{HB}} = \frac{\text{Amount}_{\text{HB}}^{\text{actual}}}{3600 \text{ Tuyere area}} \quad (8.4)$$

In particular, the injected hot blast amount ($\text{Amount}_{\text{HB}}^{\text{norm}}$) is a controlled value. The hot blast speed depends on the tuyere area and the actual hot blast amount ($\text{Amount}_{\text{HB}}^{\text{actual}}$), which in turn depends on the controlled hot blast amount ($\text{Amount}_{\text{HB}}^{\text{norm}}$), temperature, and pressure. The hot blast temperature is held at a constant value necessary for the operation of the blast furnace, and the tuyere area is also constant over large time spans. The hot blast pressure physically depends on the permeability of the burden and the hot blast parameters. It is physically not reasonable to explain the pressure of the hot blast based on the hot blast amount and speed, because the hot blast speed is a result of the pressure and the controlled hot blast amount [62].

The model for the O_2 proportion of the hot blast shown in Equation 8.5 has a very high squared correlation coefficient of 0.98. The value is a direct result of the control parameters and not particularly interesting because the value can be calculated directly using a known formula [62].

$$\begin{aligned} O_2\text{-prop}_{\text{HB}} &= \frac{c_0 \text{Temp}_{\text{HB}} + c_1 \text{Melting rate} + c_2 \text{Humidity} + c_3}{c_4 \text{Temp}_{\text{Flame}} + c_5 \text{Humidity} + c_6 \text{Heavy oil} + c_7 \text{Coal tar} + c_8} \\ &\times \frac{c_9}{c_{10} \text{Gas cons}_{\text{CO}} + c_{11} \text{Temp}_{\text{HB}} + c_{12}} + c_{13} \end{aligned} \quad (8.5)$$

The model for the flame temperature shown in Equation 8.6 has a very high squared correlation coefficient of 0.96 and contains both, the melting rate and hot blast parameters. The flame temperature can be calculated via a known formula from the O_2 proportion and temperature of the hot blast, the amount of injected materials, and a

fictional coke temperature. The coke temperature is fixed at about the same temperature of liquid slag in the furnace (about 1500°C). With this temperature it is possible to calculate the adiabatic energy balance in the flame region to determine the flame temperature [62].

The model shown in Equation 8.6 shows a statistical relation of the flame temperature to the amount of coke and the O_2 proportion of the hot blast, however, it does not match the known formula. In particular, factors that are known to have an impact on the flame temperature, namely the injected amounts of heavy oil and coal tar, have not been identified correctly [62]. Thus, symbolic regression was not able to rediscover the known physical model and the model is not useful for the blast furnace operator.

$$\begin{aligned}
 \text{Temp}_{\text{Flame}} &= c_1 \text{Melting rate} + c_2 \text{Coke} + c_3 O_2\text{-prop}_{\text{HB}} + c_4 \text{Humidity} + c_5 \\
 c_1 &= -0.24276 \\
 c_2 &= 0.41853 \\
 c_3 &= 37.35 \\
 c_4 &= -5.7866 \\
 c_5 &= 1370.8
 \end{aligned} \tag{8.6}$$

The model can be further simplified by removing the variables Coke and Melting rate. The resulting model shown in Equation 8.7 still has an R^2 value of 0.95 on the test set.

$$\begin{aligned}
 \text{Temp}_{\text{Flame}} &= c_1 O_2\text{-prop}_{\text{HB}} + c_2 \text{Humidity} + c_3 \\
 c_1 &= 37.35 \\
 c_2 &= -5.7866 \\
 c_3 &= 1448.7
 \end{aligned} \tag{8.7}$$

Equation 8.8 shows a data-based model for the total humidity with a squared correlation coefficient value of 0.86. The total humidity is physically always a result of the ambient atmospheric humidity, the enrichment of the hot blast with vapor, and the amount of water injected via tuyeres [62]. The enrichment with vapor and the amount of water injected via tuyeres are controlled by the operator. The humidity is physically not connected to the burden composition (amount of lump ore, pellets, coke) or the melting rate. Thus, the model produced by symbolic regression must be criticized as it does not match the known relationships in the actual process [62].

$$\begin{aligned}
 \text{Humidity} &= \frac{c_0 O_2\text{-prop}_{\text{HB}} + \frac{c_1}{\text{Coke}} + c_2}{c_3 \text{Temp}_{\text{Flame}} \times S} \\
 S &= c_4 \text{Coal tar} + c_5 \text{Lump ore} + c_6 \text{Melting rate} \\
 &\quad + c_7 \text{Pellets} + c_8 \text{Temp}_{\text{Flame}} + c_9 \text{Heavy oil} + c_{10}
 \end{aligned} \tag{8.8}$$

The model for the hot blast amount shown in Equation 8.9 has a squared correlation coefficient of 0.87. The model produced through data-analysis shows a connection of the hot blast amount to the melting rate and the speed of the hot blast. Additionally

the total humidity and the amount of sinter also play a role in this model for the hot blast amount. The impact of the total humidity is, however, rather small. If humidity is removed from the model by setting the numerator to 1 in Equation 8.9 the resulting output still has an R^2 of 0.85.

$$\begin{aligned} \text{Amount}_{\text{HB}} &= \frac{c_0 \text{Humidity} + c_1}{\text{Melt. rate} \times \text{Speed}_{\text{HB}} \times (c_2 \text{Melt. rate} + c_3) \times (c_4 \text{Sinter} + c_5) \times c_6} + c_7 \\ c_0 &= 2.01106 \times 10^{14} \\ c_1 &= -6.29366e \times 10^{16} \\ c_2 &= -1.5131 \\ c_3 &= 49.334 \\ c_4 &= -1.1368 \\ c_5 &= -2.355 \times 10^3 \\ c_6 &= -11.098 \\ c_7 &= 3.75 \times 10^5 \end{aligned} \tag{8.9}$$

Equation 8.11 shows a model for the melting rate with a rather high squared correlation coefficient of 0.89. The melting rate is primarily a result of the absolute amount of O_2 injected into the furnace and is also related to the efficiency of the furnace. A simple approximation for the melting rate is

$$\frac{\text{Total amount of } O_2}{[220 \dots 245]} \tag{8.10}$$

When the furnace is working properly the melting rate is higher ($O_2/220$), when the furnace is working inefficiently the melting rate decreases ($O_2/245$) and high cooling losses can be observed. Additional factors that are known to effect the melting rate are the burden composition and the amount of slag. Data-based modeling of the melting rate is interesting because the physical causes for changes in the melting rate are not fully understood and a data-based model could lead to better insights into the influencing factors. The model generated through data-analysis shown in Equation 8.11 also shows the known relation of the melting rate and the amount of O_2 . The cooling losses and the amount of lump ore have also been identified as factors connected to the melting rate. Additionally, the gas utilization of CO plays a role in this model for the melting rate.

$$\begin{aligned} \text{Melting rate} &= \log(c_0 \times \text{Temp}_{\text{HB}} \times O_2\text{-prop}_{\text{HB}} \times (c_1 \text{Cool. loss} + c_2 \text{Amount}_{\text{HB}} + c_3) \\ &\quad + c_4 \times \text{Gas util}_{\text{CO}} \times (c_5 \text{Lump ore} + c_6) \times (c_7 \text{Amount}_{\text{HB}} + c_8)) \end{aligned} \tag{8.11}$$

The model for slag shown in Equation 8.12 has a squared correlation coefficient of 0.77. It is possible to calculate the amount of slag directly from the amounts of input material charged at the top of the blast furnace, thus the data-based model for the amount of slag is not useful [62].

$$\text{Slag} = c_0 \text{Sinter} + \frac{c_1 \text{Heavy oil} + c_2}{c_3 \log(\text{Alkali}) + c_4} + c_5 \text{Alkali} \quad (8.12)$$

8.1.4. Concluding Remarks

Using a comprehensive symbolic regression approach a number of data-based models have been identified for certain variables in the blast furnace process that approximate the observed values rather accurately.

In the detailed analysis of the models presented in this section it became clear, that the majority of the models are not useful for the blast furnace operator because the real process is not described correctly [62]. In particular, the symbolic regression models often contain input variables which are known to be independent from the target variable, whereas input variables which do have an influence on the target variable are not identified correctly.

Additionally, the comprehensive symbolic regression approach produced many models which are not useful for the blast furnace operator because the modeling process is untargeted. In particular, some variables which are considered as target variables are not relevant for the analysis of the process because they are directly or indirectly controlled by the blast furnace operator [62]. Also, the value of some variables can be determined through known formulas which are more accurate than the data-based models.

As described in the previous sections, many variables in the blast furnace process are implicitly related, either because of underlying physical relations, or because of the external control of blast furnace parameters. This causes problems in the identification and description of symbolic regression models, because it is possible to express an identified dependency in multiple different but semantically identical ways via the implicit relations. Usually, implicit relations are not known a-priori in data-based modeling methods, thus a method to identify all implicit dependencies can be useful. However, in the situation of the blast furnace process many of the implicit dependencies are already known, thus the untargeted approach is inefficient and misleading. Instead, it is better to create models for a few carefully selected target variables, where the set of input variables is restricted to a few variables. For the definition of the modeling scenarios, as well as for the interpretation of the models, expert knowledge about the process and the implicit dependencies is necessary.

In retrospect the comprehensive symbolic regression approach is not useful for modeling the blast furnace process, where many physical relations are known beforehand. A very interesting extension of that approach, left for future work, would be to include a-priori knowledge into the modeling process. If the control parameters and known causal chains are predefined and supplied as input to the modeling approach it would be possible to restrict the search for actually relevant and potentially interesting models.

8.2. Econometric Modeling

Macro-economic models describe the dynamics of economic quantities of countries or regions, as well as their interaction on international markets. Macro-economic variables that play a role in such models are for instance the unemployment rate, gross domestic product, current account figures and monetary aggregates. Macro-economic models can be used to estimate the current economic conditions and to forecast economic developments and trends. Therefore macro-economic models play a substantial role in financial and political decisions.

Genetic programming has been used to rediscover well known non-linear econometric models from noisy datasets [100, 101]. Numerous papers have been published, describing applications of genetic programming in finance and econometrics, in particular for modeling and prediction of financial time series [36, 121, 92, 216, 144, 93, 31] and the discovery of effective trading rules [143, 141, 142, 140, 131, 50, 51, 237].

In this contribution we take up the idea of using symbolic regression to generate models describing macro-economic interactions based on observations of economic quantities. However, contrary to the constrained situation studied in [100], we use a more extensive dataset with observations of many different economic quantities, and aim to identify all potentially interesting economic interactions that can be derived from the observations in the dataset. Also, we are not concerned about the prognosis of economic time series, but in the relation of different economic variables over the observed time span.

8.2.1. Macro-Economic Dataset

The dataset we used for the discovery of macro-economic models has been collected and kindly provided to us by Dr. Stefan Fink. The original dataset contains monthly observations of 104 economic variables and indexes from the United States of America, Germany, and the Euro zone in the time span from January 1978 until December 2008. However, not all variables contain values for the full time span so, as preparation for modeling with GP, we reduced the number of rows to cover the time span from January 1980 until July 2007 (331 rows). Next we reduced the set of input variables and kept only those variables, for which values for the whole time span are available. All variable values have been scaled to values between 1 and 2. The variable days is a index variable for the number of days elapsed since 01/01/1980. Using this variable, the GP process can theoretically identify seasonal changes in the variables. Table 8.5 lists the remaining variables of the macro-economic dataset.

Preliminary analysis of the dataset showed, that some of the variables are strongly correlated. The strongly correlated variables pairs are: the number of housing starts and the number of building permits, the ISM manufacturing purchase managers index and the Chicago purchase managers index, and the university of Michigan sentiment index and the university of Michigan conditions index. The three variables correlated variables, housing starts, ISM mfg PMI, and U. Mich. sentiment have been removed from the dataset.

The following variables have a general rising trend and are pairwise strongly cor-




























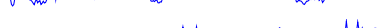









Variable	Data
Days	
d(US Average Earnings)	
US Building Permits	
US Capacity utilization	
US Chicago PMI	
US Consumer Confidence	
d(US Consumption, adj)	
d(US CPI inflation, mm)	
d(US Current Account)	
US Domestic Car Sales	
d(US Existing Home Sales)	
US Fed Budget	
US Foreign Buying, T-Bonds	
US Help Wanted Index	
US Housing Starts	
d(US Industrial Output mm)	
d(US International Trade \$)	
US ISM mfg PMI	
d(US Leading Indicators)	
US Manufacturing Payroll	
US Mich conditions prel	
US Mich expectations prel	
US Mich InflErw 1y	
US Mich InflErw 5y	
US Mich sentiment prel	
US National Activity Index	
US New Home Sales	
d(US nonfarm payrolls)	
d(US Personal Income)	
US Phil Fed business Index	
d(US Producer Prices mm)	
US Unemployment	
d(EZ Producer Price mm)	
d(GER Exports)	
d(GER Imports mm)	
d(GER Producer Price)	
GER Wholesale price index	

Table 8.5.: List of economic variables considered for the identification of macro-economic relations

related: personal income, existing house sales, current account, international trade, consumption, leading indicators, industrial output, average earnings, non-farm payrolls, producer price, CPI inflation, imports (GER), exports (GER), producer price (EZ). These correlations are not particularly useful, so we decided to study the derivatives (monthly changes) of the variables instead of the absolute values to find more interesting relations. The derivative variables ($d(x)$) are calculated using the five point formula for the numerical approximation of the derivative (see Section A.2). The drawback of the numerical approximation of the derivative is, that the noise in the original values is amplified by the calculation of the derivative and it is more difficult to produce accurate models for the derivative variable.

8.2.2. Modeling

The dataset is split into training (rows 13–300) and test (rows 300–331). The testing period is January 2005 to July 2007. Only rows 13–200 are used for fitness evaluation and rows 100–300 are used as internal validation set for overfitting detection and for the selection of the final (best on validation) model. Note, that there is an overlapping region of rows that are used for the fitness calculation and for internal validation. The dataset is rather small and the overlap is on purpose to increase the number of rows, used for validation. The drawback of the overlap is, that the chance to return overfit models is larger.

The goal of the modeling step is to identify the network of relevant variable interactions in the macro-economic dataset. Thus, several symbolic regression runs were executed to produce approximation models for each variable of the dataset. The same parameter settings were used for all runs. Only the target variable and the list of allowed input variables was adapted. The GP parameter settings for our experiments are specified in Table 8.6. We used rather standard GP configuration with tree-based solution encoding, tournament selection, sub-tree swapping crossover, and two mutation operators. The fitness function is the squared correlation coefficient of the model output and the actual values of target variables. Only the final model is linearly scaled to match the location and scale of the target variable [95].

In the experiments we used two adaptations of the algorithm to reduce bloat and overfitting described in the previous chapters. Dynamic depth limits [188] with an initial depth limit of seven are used to reduce the amount of bloat. An internal validation set is used to reduce the chance of overfitting. Each solution candidate is evaluated on the training and on the validation set. Selection is based solely on the fitness on the training set, the fitness on the validation set is used as an indicator for overfitting. Models that have a high training fitness but a low validation fitness are likely to be overfit. Thus, after each iteration the correlation of the training- and validation fitness values of all solution candidates in the population is calculated using Spearman’s rank correlation $\rho(\text{Fitness}_{\text{train}}, \text{Fitness}_{\text{val}})$. If the correlation of training- and validation fitness in the population drops below a certain threshold the algorithm is stopped.

Parameter	Value
Population size	2000
Max. generations	150
Parent selection	Tournament (group size = 7)
Replacement	1-Elitism
Initialization	PTC2 [127]
Crossover	Sub-tree-swapping
Mutation	7% One-point, 7% sub-tree replacement
Tree constraints	Dynamic depth limit (initial limit = 7)
Model selection	Best on validation
Stopping criterion	$\rho(\text{Fitness}_{\text{train}}, \text{Fitness}_{\text{val}}) < 0.2$
Fitness function	R^2 (maximization)
Function set	+, -, *, /, avg, log, exp, sin
Terminal set	constants, variables, lagged variables (t-12) ... (t-1)

Table 8.6.: Genetic programming parameters.

8.2.3. Results

For each variable of the dataset 30 independent GP runs have been executed using the open source software HeuristicLab on four blades of a blade-system each equipped with an 8-core Intel Xeon processor and 32GB RAM. The result is a collection of 990 models (generated in the same number of GP runs) representing the (non-linear) interactions between all variables. Figure 8.5 shows the box-plot of the squared correlation coefficient (R^2) of the model output and the original values on the test set for the 30 models for each variable.

Variable Interaction Network

For each target variable we calculated the frequency-based relevance of all possible input variables. In Figure 8.6 the three most relevant input variables for each target variable are shown, where an arrow ($a \rightarrow b$) means that variable a is a relevant variable in models for variable b . The variable pairs with large correlations ($R^2 > 0.85$) have been grouped. Only a single variable of each group was allowed as input variable. The reasoning behind this is, that if there is a high correlation between variables, the variables can be used as input interchangeably.

The network of relevant variables shows many strong double-linked variable relations. Strongly related variables discovered by GP are for instance *exports* and *imports* of Germany, *consumption* and *existing home sales*, *building permits* and *new home sales*, *Chicago PMI* and *non-farm payrolls* and a few more. GP also discovered a chain strongly related variables, connecting the *producer price indexes* of the euro zone, Germany and the US with the *US CPI inflation*.

A large strongly connected cluster that contains the variables *unemployment*, *capacity utilization*, *help wanted index*, *consumer confidence*, *U. Mich. expectations*, *U. Mich. conditions*, *U. Mich. 1-year inflation*, *building permits*, *new home sales*, and *manufacturing payrolls* has also been identified by our approach.

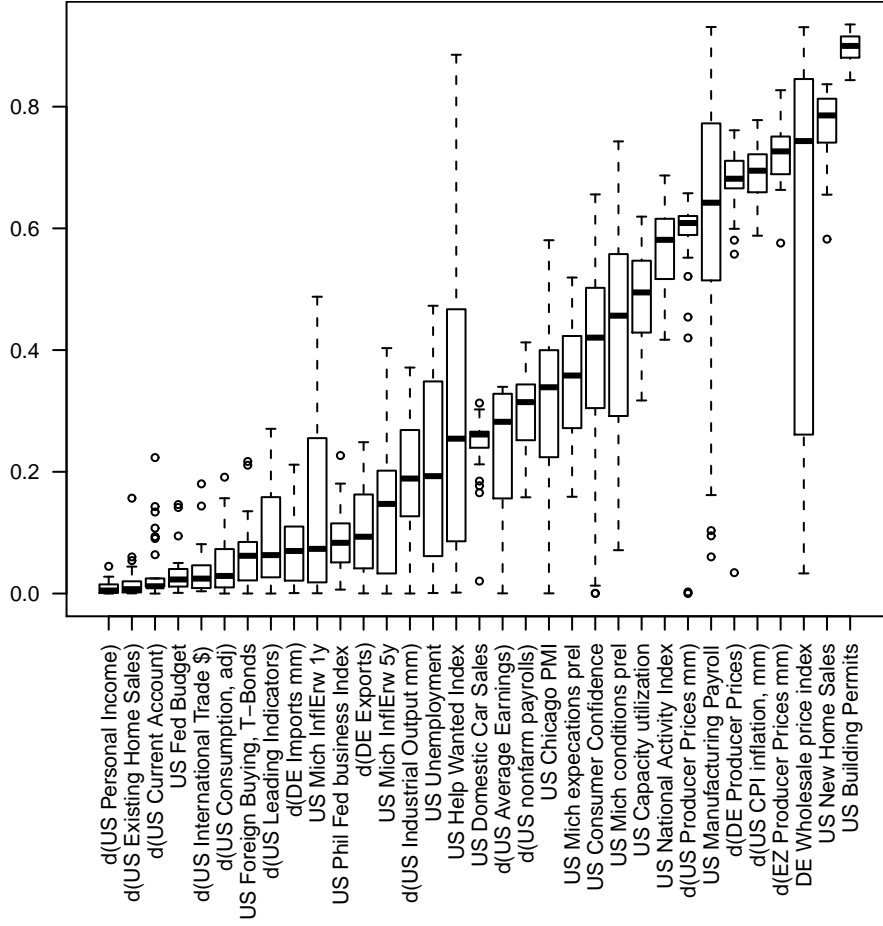


Figure 8.5.: Box-plot of R^2 values on the test set of models for the macro-economic dataset.

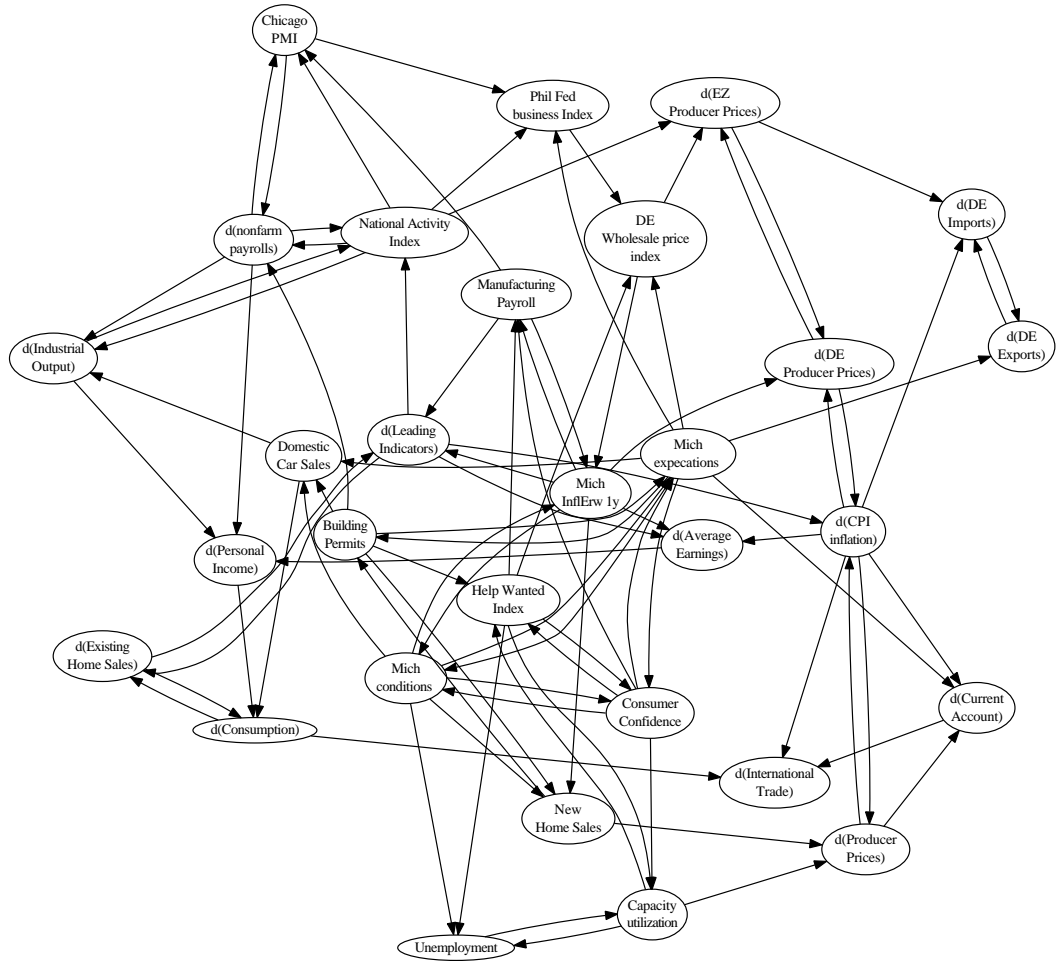


Figure 8.6.: Relationships of macro-economic variables identified with comprehensive symbolic regression and frequency-based variable relevance metrics. This figure has been plotted using GraphViz.

Variable	Model	R^2
Building permits	(8.13)	0.93
New home sales	(8.14)	0.82
Manufacturing payrolls	(8.15)	0.89
Help wanted index	(8.16)	0.82
Wholesale price index (GER)	(8.17)	0.61
Producer price index (GER)	(8.20)	0.74
Producer price index (EZ)	(8.18)	0.82
CPI inflation	(8.19)	0.93
Capacity utilization	(8.21)	0.76
National activity index	(8.22)	0.66
U. Michigan conditions index	(8.23)	0.69

Table 8.7.: Overview model accuracy for the simplified macro-economic models identified by GP.

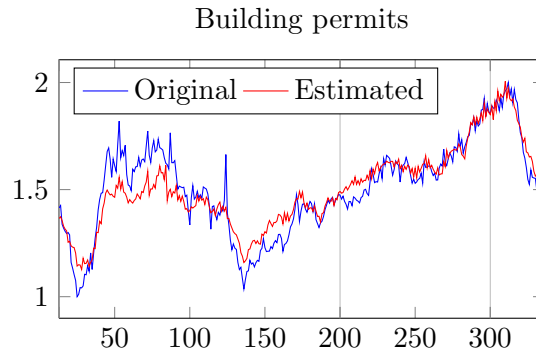
Outside of the central cluster, the variables *national activity index*, *CPI inflation*, *non-farm payrolls* and *leading indicators* also have a large number of outgoing connections, indicating that these variables play an important role for the approximation of many other variables.

Detailed Models

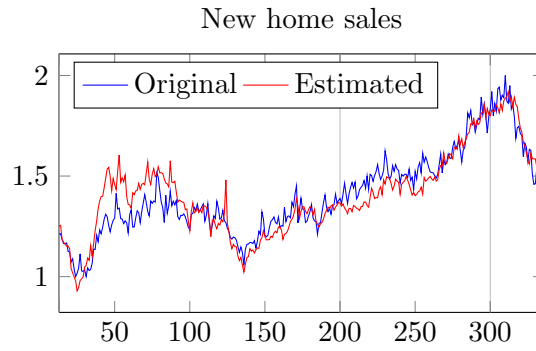
The variable interaction network only provides a course grained high level view on the identified macro-economic interactions. To obtain a better understanding of the identified macro-economic relations it is necessary to analyze single models in more detail. In the following we present a number of manually selected and simplified models identified by GP. Table 8.7 shows an overview of model accuracy on the test set for the presented models.

The help wanted index is calculated from the number of job advertisements in major newspapers and is usually considered to be related to the unemployment rate [40], [1]. The model for the *help wanted index* shown in Equation 8.16 has a R^2 value of 0.82 on the test set. The model includes the *manufacturing payrolls* and the *capacity utilization* as relevant factors. Interestingly, the unemployment rate which was also available as input variable is not used, instead other indicators for economic conditions (Chicago PMI, U. Mich cond.) are included in the model. Interestingly the model also includes the *building permits* and *wholesale price index of Germany*.

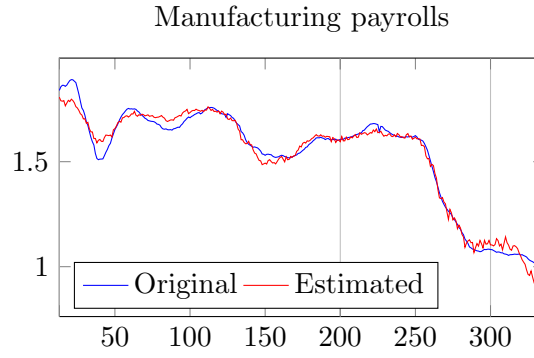
$$\begin{aligned}
\text{Help wanted index} = & \text{Building permits} + \text{Mfg payroll} \\
& + \text{Capacity utilization} + \text{Wholesale price index (GER)} \\
& + \text{Chicago PMI} + \text{Mfg Payroll}(t - 5) \\
& + \text{Mich cond.}(t - 3)
\end{aligned} \tag{8.16}$$



$$\begin{aligned}
 \text{Building permits} = & \text{Unemployment} + \text{Unemployment}(t - 5) \\
 & + \text{New home sales} \\
 & + \text{New home sales}(t - 2) + \text{New home sales}(t - 4) \\
 & + \log(\text{Consumer conf.}(t - 1)) + \text{Domestic car sales}
 \end{aligned} \tag{8.13}$$



$$\text{New home sales} = \text{Building permits} + \frac{1}{\log(\text{mfg payroll}) + c_0} \tag{8.14}$$



$$\begin{aligned} \text{Mfg payrolls} = & \frac{\text{Days} + c_0}{\text{Help wanted index} \times \text{Help wanted index}(t - 6)} \\ & + \log(\text{Consumer conf.}(t - 1) + \text{Mich 1y infl}(t - 6)) \end{aligned} \quad (8.15)$$

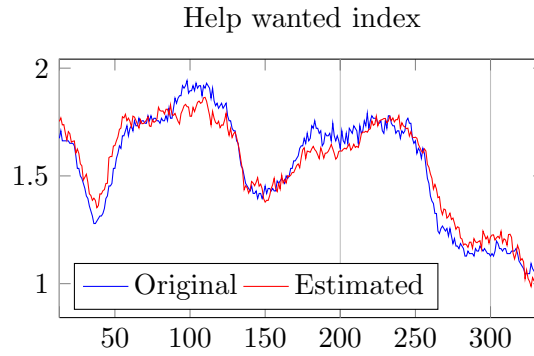
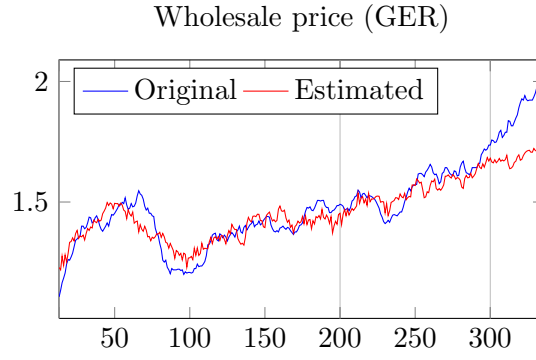


Figure 8.7.: Line chart of the actual value of the *US Help wanted index* and the estimated values produced by the model (Equation 8.16). Test set starts at index 300.

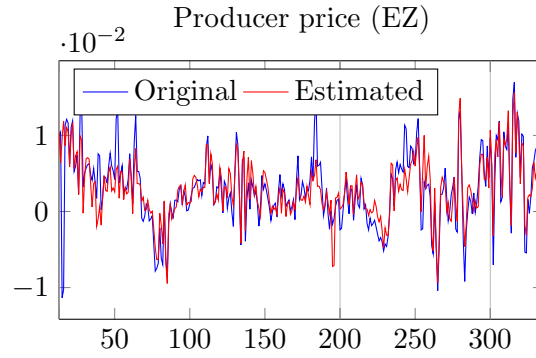
Figure 8.7 shows a line chart for the actual values of the *help wanted index* in the US and the estimated values of the model (Equation 8.16) over the whole time span covered by the dataset.

The *consumer price index* measures the change in prices paid by customers for a certain market basket containing goods and services, and is measure for the inflation in an economy. The output of the model for the *CPI inflation* in the US shown in Equation 8.19 is very accurate with a squared correlation coefficient of 0.93 on the test set. The model approximates the *consumer price index* based on the *unemployment*, *car sales*, *New home sales*, and the *consumer confidence*.

$$\begin{aligned} \text{CPI inflation} = & \text{Unemployment} + \text{Domestic car sales} + \text{New home sales} \\ & + \log(\text{New home sales}(t - 4) + \text{New home sales}(t - 2)) \\ & + \text{Consumer conf.}(t - 1) + \text{Unemployment}(t - 5)) \end{aligned} \quad (8.19)$$



$$\begin{aligned}
 \text{Wholesale price (GER)} = & \text{Help wanted index} + \text{Consumer conf.} \\
 & + \exp(\text{Help wanted index} + \text{Mich expect.}(t-1)) \\
 & + \frac{1}{\text{Mich expect.}(t-8)} + \text{Capacity util.}(t-2) \\
 & + \exp(\text{Help wanted index}(t-12)) \\
 & \times (\text{Help wanted Index} + c_0)
 \end{aligned} \tag{8.17}$$



$$\text{Producer price (EZ)} = d(\text{CPI inflation}) + d(\text{Producer price (GER)}) \tag{8.18}$$

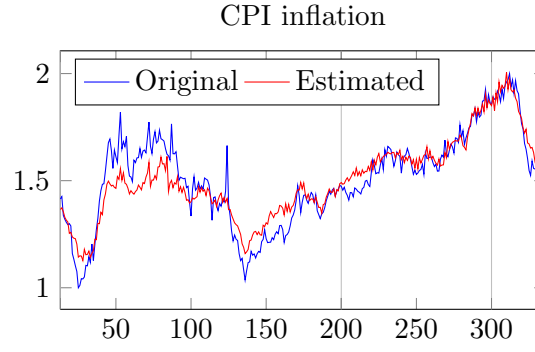
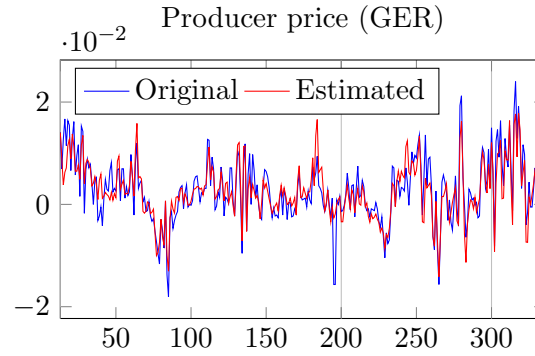


Figure 8.8.: Line chart of the actual value of the *US CPI inflation* and the estimated values produced by the model (Equation 8.19).

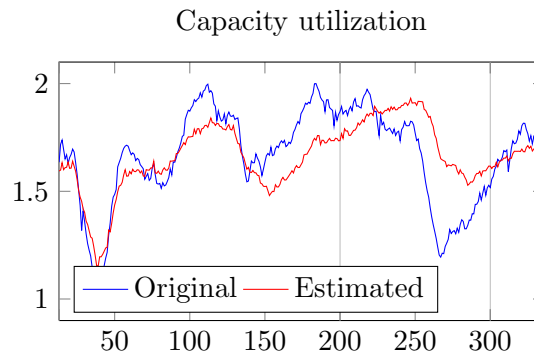


$$\begin{aligned}
 \text{Producer price (GER)} = & d(\text{Producer Price (EZ)}) + \text{Mich 1y infl.}(t - 6)^2 \\
 & + d(\text{EZ Producer price}) \times \text{Consumer conf.} \\
 & + d(\text{EZ Producer price}) \times \text{Mich cond.}(t - 8))
 \end{aligned} \tag{8.20}$$

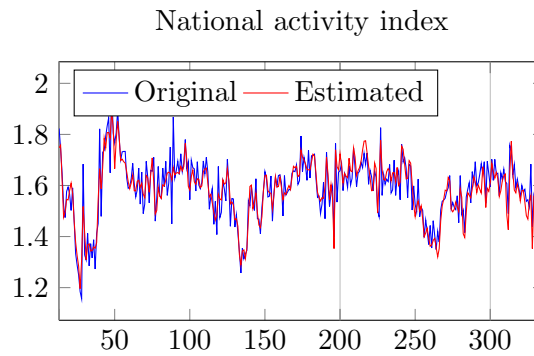
Figure 8.8 shows a line chart for the actual values of the *CPI inflation* in the US and the estimated values of the model (Equation 8.19) over the whole time span covered by the dataset. Notably the drop of the CPI in the test set (starting at index 300) is estimated correctly by the model.

8.2.4. Concluding Remarks

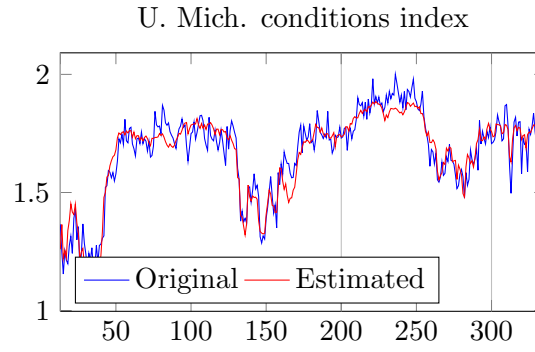
The application of the proposed approach on the macro-economic dataset resulted in a high level overview of macro-economic variable interactions. The variable interaction network provides information that is not apparent from analysis of single models, and thus supplements the information gained from detailed analysis of single models. In the experiments we used dynamic depth limits to counteract bloat and an internal validation set to detect overfitting using the correlation of training- and validation fitness. A number of selected models for instance for the *US Help wanted index* and the *US CPI*



$$\text{Capacity utilization} = \text{Unemployment} + \text{Mfg payrolls} \quad (8.21)$$



$$\begin{aligned} \text{National activity} = & (d(\text{Non-farm payrolls}) + d(\text{Non-farm payrolls})(t-1) \\ & + d(\text{Industrial output}) + d(\text{Industrial output})(t-1) \\ & + \text{Building permits}) \\ & \times \left(\frac{1}{d(\text{Existing home sales})(t-4)} \right. \\ & \left. + \text{New home sales}(t-12) + c \right) \end{aligned} \quad (8.22)$$



$$\begin{aligned}
 \text{Mich. cond.} &= \text{Mfg payrolls}(t - 10) \\
 &\times \text{Consumer conf.} \\
 &\times (\text{Mich 1-year inflation} + \text{Mich expect.} \\
 &\quad + \text{Help wanted index}(t - 2) + c) \\
 &\times \left(\frac{c}{\text{Consumer conf.} \times (\text{Mich expect.} + c)} \right. \\
 &\quad \left. + \text{Help wanted index} \times \text{Wholesale price (GER)}(t - 9) + c \right) \\
 &\times \left(\frac{c}{\text{Consumer conf.} \times (\text{Mich expect.} + c)} + c \right)
 \end{aligned} \tag{8.23}$$

inflation have been presented and discussed in detail. The models are rather accurate also on the test set and are relatively comprehensible.

Correlated variables

X33, x_{51} , x_{52} , x_{53}
X35, x_{37}
X31, x_{47}
X39, x_{40}
X1, x_7 , x_{10} , x_{38}
X4, x_{20} , x_{21}
X54, x_{55}
X9, x_{16} , x_{17} , x_{18} , x_{19}
X23, x_{24} , x_{25} , x_{26} , x_{27} , x_{28}

Table 8.8.: Clusters of strongly correlated variables in the chemical dataset. The variable used as representative for the cluster in the modeling process is indicated in bold font.

8.3. Chemical Process

The last application studied in this chapter is the analysis of data from a chemical process. The dataset analyzed in this section has been prepared and published by Arthur Kordon of Dow Chemical for the symbolic regression competition, which has been held as a side event of the EvoStar conference 2010. Only little information about chemical process, from which the data was collected, is available. The measurements stem from a real industrial process. The input variables are process parameters like temperatures, pressures and material flows (input material). The target variable y describes the chemical composition of the output of the process. The values of y are measured in the laboratory and are noisy and rather expensive to measure. The objective is to find a robust model which accurately approximates the lab value y from the process parameters (virtual sensor). Through interpretation of the model it is possible to gain a better understanding of the impacts of different process parameters on the chemical composition. This knowledge can be used for controlling the process, in order to improve product quality.

Some of the input variables are implicitly related so a full search for non-linear relations on the whole dataset is beneficial in order to get a better understanding of the whole process. This is also helpful for the interpretation of model for y as the implicit relations can be used to generalize over multiple equally accurate models for y .

8.3.1. Modeling

Preliminary analysis shows that a number of variable pairs are strongly correlated leading to a number of clusters of strongly correlated variables. For each cluster only a single representative variable is kept in the dataset the remaining variables in the cluster are removed because they can be expressed through the representative variable of the cluster. Table 8.8 lists the clusters identified by correlation analysis.

The dataset contains 57 input variables x_1 – x_{57} and the target variable y . The dataset

Parameter	Value
Population size	2000
Max. generations	150
Parent selection	Tournament
	Group size = 7
Replacement	1-Elitism
Initialization	PTC2
Crossover	Sub-tree-swapping
Mutation rate	15%
Mutation operator	One-point
	One-point constant shaker
	Sub-tree replacement
Tree constraints	Dynamic depth limit
	Initial limit = 7
Model selection	Best on validation
Stopping criterion	$\text{Corr}(\text{Fitness}_{\text{train}}, \text{Fitness}_{\text{val}}) < 0.2$
Fitness function	R^2 (maximization)
Function set	+, -, *, /, avg, log, exp
Terminal set	constants, variables

Table 8.9.: Genetic programming parameters for the symbolic regression experiments with the chemical dataset.

is split into a training partition (747 rows) and a test partition (319 rows). The first 30 rows of the training partition are not used at all, rows 30–388 are used to calculate fitness and rows 388–746 are used as internal validation partition for overfitting detection and to select the final (validation-best) model. Fitness is calculated as squared correlation coefficient of the model output and the original target values. The GP parameter settings are specified in Table 8.9. The final model is linearly scaled using the target values from row 30–746. The squared correlation coefficient R^2 values reported are calculated from the final model output applied to the test partition.

8.3.2. Results

Figure 8.9 shows the squared correlation coefficient (R^2) over 30 models for each variable. GP reliably found very accurate models for the variables x_9 (representative for x_{16} , x_{17} , x_{18} , and x_{19}), x_1 (representative for x_7 , x_{10} , and x_{38}), x_{46} , and x_{36} . The R^2 of the models for x_{54} and x_{23} are scattered over a large range but a few GP runs produced accurate models. The models for the target variable y have a median R^2 of 0.6 on the test set. For variable x_4 (which is the representative of a cluster also containing x_{20} and x_{21}) no accurate models have been found.

Figure 8.10 shows the network of the most relevant input variables for modeling each variable. The three most relevant input variables, as determined by the frequency-based relevance metric are shown for each variable. The variable relation network shows that variables x_{11} and x_{12} are also strongly connected to the cluster with the representative x_{23} . There is a long chain of clusters strongly connected to the target variable y . The

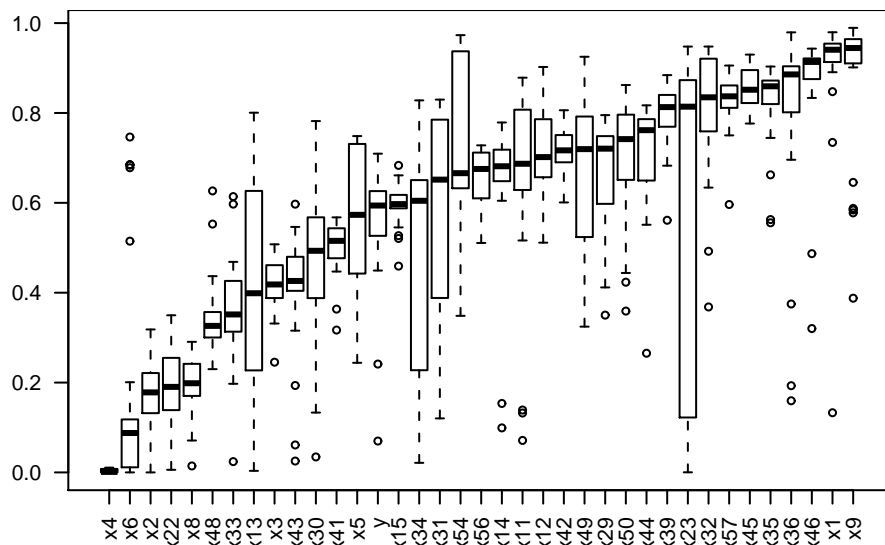


Figure 8.9.: Box-plot of R^2 value on the test set of models for the chemical dataset.

target variable y is connected to the cluster with representative x_{35} which is in turn strongly connected to the cluster with representative x_1 which is again strongly connected to the cluster with representative x_{54} which is strongly connected to the cluster with representatives x_9 . The most relevant input variables for the target variable y are x_{35} , x_{49} and x_{32} (which is itself connected strongly to x_{49}). Interestingly the variable y plays an important role for the approximation of the cluster with representative x_{33} , however, as shown in the box-plot the models for x_{33} are not very accurate. Generally in the chemical dataset, there are many variable pairs that are doubly-linked, which is an indicator that the variable pair is strongly related.

Central variables of the network with many outgoing arrows are: x_{13} , x_8 , x_6 , x_5 , x_{30} , and x_{42} .

The variables x_{11} , x_{12} , and the cluster with representative x_{23} are not among the most relevant variables for modeling any other variable. The same is true for the cluster containing variables x_4 , x_{20} , and x_{21} .

8.3.3. Detailed Results

In the following a number of selected and simplified models are presented in more detail. Table 8.10 lists all models together with the squared correlation coefficient of the model output and the original target values on the test set. The unguided search for models in the chemical dataset produced very accurate linear models for x_{31} (8.28), x_{32} (8.29), x_{46} (8.33), and x_{49} (8.35). The model for the target variable y contains x_{32} , x_{31} , and x_{49} as input variables, so further information about the nature of these two variables is relevant for the interpretation of the model produced for y . For the variable x_{35} , that

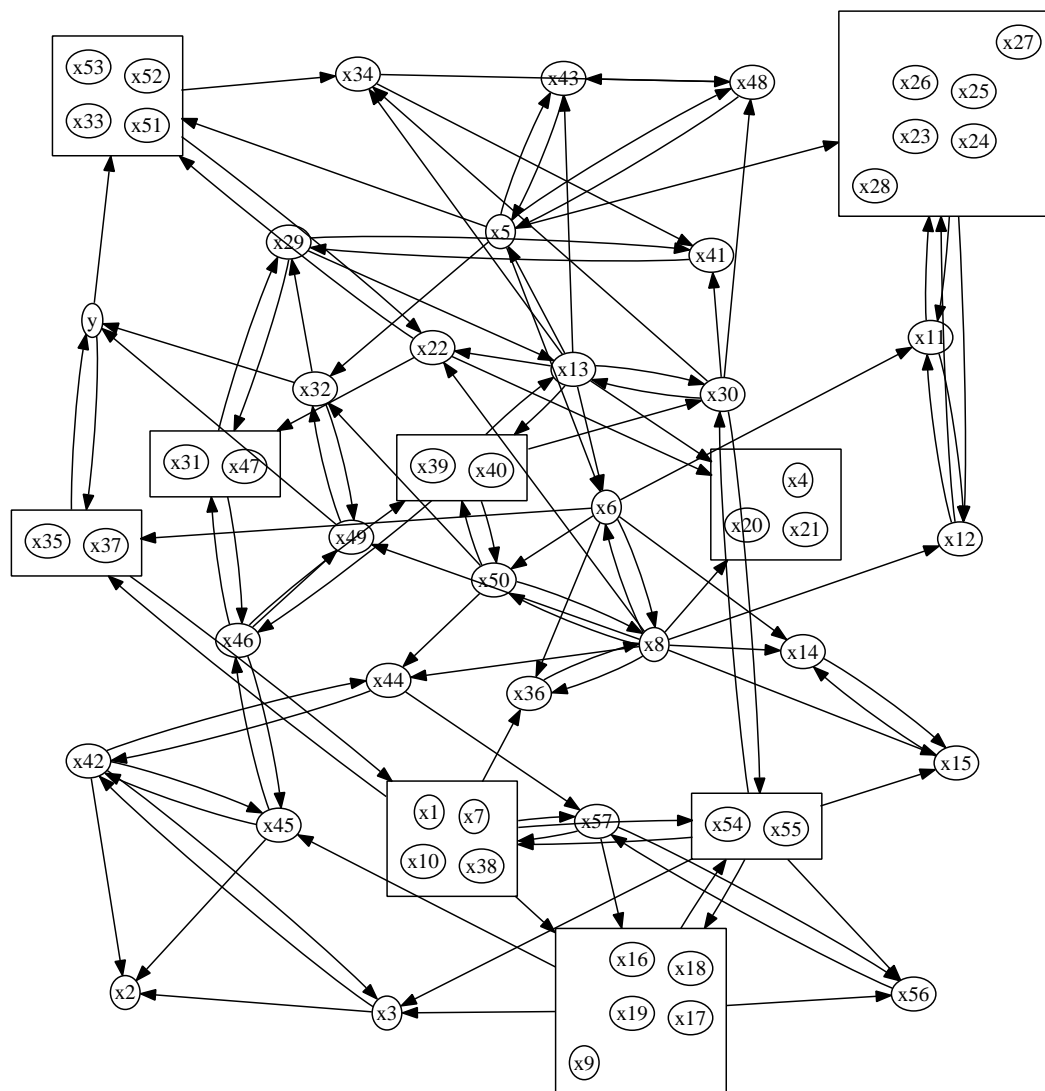


Figure 8.10.: Relationships of variables in the Chemical-I dataset identified with unguided GP-based data mining

Variable	Model	R^2
x_1	(8.24)	0.98
x_9	(8.25)	0.85
x_{14}	(8.26)	0.76
x_{15}	(8.27)	0.65
x_{31}	(8.28)	0.83
x_{32}	(8.29)	0.96
x_{35}	(8.30)	0.90
x_{39}	(8.31)	0.86
x_{42}	(8.32)	0.84
x_{46}	(8.33)	0.93
x_{48}	(8.34)	0.51
x_{49}	(8.35)	0.93
x_{54}	(8.36)	0.71
y	(8.3.3)	0.64

Table 8.10.: Overview of models for the chemical dataset.

also occurs in the model for y , a very accurate non-linear model has been found (8.30) which relates x_{35} to variables x_6 , x_{33} , and x_{57} . The only variable occurring in the model for y , which cannot be approximated accurately through other variables, is x_{48} (8.34).

$$x_1 = x_8 + x_9 + x_9^2 + x_{15} + x_{34} + x_{35} + x_{45} + y + \frac{x_9(x_{57} + x_{34} + x_{49})}{x_{54}} \quad (8.24)$$

$$x_9 = \frac{1}{x_{35} + x_{54}} \quad (8.25)$$

$$x_{14} = x_{15} + x_{50} + \frac{1}{\exp(x_8)} \quad (8.26)$$

$$x_{15} = \log \left(\frac{\frac{\exp(x_{14})}{\log(y)} + c_0}{\log(x_5 + c_1) \times \exp(x_{14} + c_2)} \right) \quad (8.27)$$

$$x_{31} = x_{30} + x_{32} + x_{46} + x_{50} \quad (8.28)$$

$$x_{32} = x_5 + x_{14} + x_{31} + x_{33} + x_{44} + x_{46} + x_{49} + x_{50} \quad (8.29)$$

$$x_{35} = \frac{1}{x_{33} + \frac{x_1^2(x_6+c_0)}{x_{57}+c_1} + c_2} \quad (8.30)$$

$$x_{39} = \frac{x_{30} + \frac{x_{13}}{(x_{50}+c_0)^2} + c}{x_5 + x_{29} + x_{41} + x_{45} + x_{46} + x_{50} + c} \quad (8.31)$$

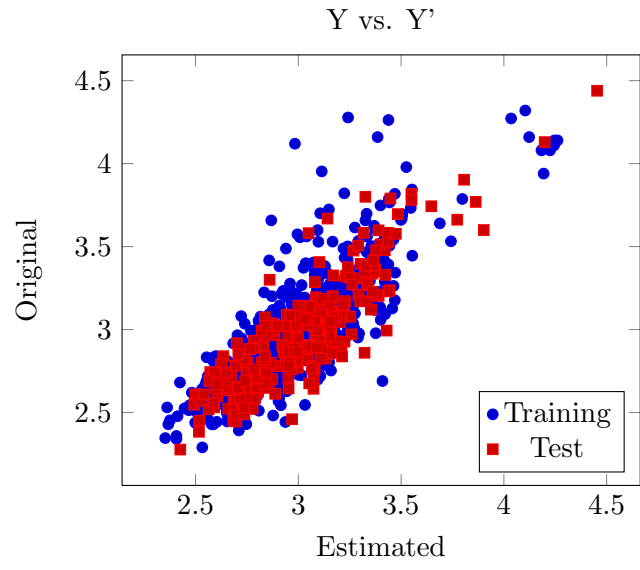
$$x_{42} = \frac{1}{x_{34} + x_{43} + x_{45}} \quad (8.32)$$

$$x_{46} = x_3 + x_5 + x_8 + x_{30} + x_{31} + x_{36} + x_{39} + x_{44} + x_{45} + x_{49} \quad (8.33)$$

$$x_{48} = \frac{(x_5 + x_{44} + c)(x_{30} + x_{33} + x_{34} + c)(x_{33} + c)(x_{42} + c)}{x_{34} + c} \quad (8.34)$$

$$x_{49} = x_1 + x_8 + x_{31} + x_{32} + x_{44} + x_{46} + x_{54} + x_{50} \quad (8.35)$$

$$x_{54} = \exp\left(\exp\left(\frac{x_9}{x_{22}x_{30}}\right)\right) \quad (8.36)$$



$$y = x_{31} + x_{32} + x_{49} + \frac{x_{48} + c_0}{x_{35}} + \frac{\log(x_{49}) + c_1}{\log(x_{32}) + c_2} \quad (8.37)$$

9. Concluding Remarks

The main topic of this work is the application of genetic programming, in particular symbolic regression, for the identification of comprehensible and accurate models in real-world scenarios. We have focused on aspects of symbolic regression that are relevant for practical applications. The main contribution is the description of a comprehensive symbolic regression method for the identification of variable interaction networks. Such networks provide a coarse grained overview of the strongest (non-linear) dependencies of variables in the studied system. This method is based on a new approach to quantify relevant input variables, which is presented for the first time in this thesis.

In practical applications a major concern is the comprehensibility and generality of models. Thus, methods to reduce bloat and overfitting are prominently featured in this thesis as well. Additionally, we also describe a new method for auto-regressive modeling of multi-variate time series with genetic programming. The methods described in the first part of this thesis are applied to a number of real-world problems in the second part.

The following topics are covered in this thesis:

- The problem of bloat and a number of bloat control methods are discussed. In this thesis the effect of offspring selection on bloat is analyzed for the first time. We have observed that offspring selection does not reduce bloat, instead it has the tendency to increase bloat because of the higher selection pressure.
- Overfitting is an issue in symbolic regression that can occur even in the absence of bloat. Bloat and overfitting are two related but separate phenomena. We suggest that an effective symbolic regression approach should include specific countermeasures for bloat and overfitting. In this thesis a new approach for the detection of overfitting based on the correlation of training- and validation fitness is described for the first time. Based on this overfitting criterion, we propose two new approaches to reduce overfitting. The first approach uses covariant parsimony pressure to adaptively control the average program length in the population based on the overfitting criterion. The second approach uses selective pruning in overfitting phases to remove code fragments that have only a small effect on the fitness of the model but potentially have a negative effect on the generalization ability of the model.

The newly proposed overfitting criterion and the two approaches for overfitting reduction are tested on two real world datasets where standard genetic programming produces overfit models and based on the results we have observed that the overfitting criterion reliably indicates overfitting phases and that the two proposed methods reduce overfitting.

In this thesis the effects of offspring selection on overfitting are analyzed for the first time. We observe that the overfitting effect is stronger because of the higher selection pressure resulting from offspring selection. Thus, we strongly advise to use an effective method for the validation of symbolic regression models produced in runs with offspring selection.

- Genetic programming has the tendency to build convoluted solutions containing unoptimized code or introns. This effect can be reduced but not completely prevented through bloat control methods. Such convoluted solutions are problematic in practical applications, thus we suggest to simplify symbolic regression solutions in two steps. First the model should be simplified using algebraic transformations and in a second step the branches which have only a minor effect on the output should be removed through either automatic or manual pruning. In this thesis we show that this simplification approach significantly improves the comprehensibility of symbolic regression solutions while the fitness of the model is only slightly decreased.

In this context, we also suggest that a small function set including only few simple operators should be preferred for practical applications. Arithmetic expressions can be simplified easily and are easier to comprehend than expressions with deeply nested complex functions.

- Measuring variable importance is a big topic in regression modeling and is connected to feature selection. In practical applications the information about the relevant factors that might have an effect on a target variable is often very valuable.

In this thesis we review different ways to determine relative variable importance in regression modeling and propose two new methods to determine relevant variables in symbolic regression. The first method is specific for symbolic regression, and calculates variable importance as the average relative frequency of variable references over the whole symbolic regression run. The second method is derived from the variable importance metric used for random forests and uses permutation sampling to estimate the relative importance of a variable in a regression model.

The different methods are applied to a number of benchmark problems where the variable importance is known a-priori, and the results show that the permutation sampling approach produces the best approximations for the variable importance.

Additionally, we discuss at length an approach to determine variable importance, where the values of the variable are replaced by the average value. We conclude that this approach should not be used as it produces misleading variable importance results even for very simple examples.

- Real-world datasets are often characterized by many highly correlated variables and other non-linear variable interactions. In this thesis the approach of comprehensive symbolic regression is described for the first time. The approach aims to produce a coarse-grained overview of all variable dependencies that are apparent in a dataset

and is based on variable relevance metrics. We suggest to apply the method to explore implicit dependencies in the dataset whenever a new data-analysis problem is approached. The advantage of this method is that the implicit relations, which might be known by a domain expert but are often unknown to the data analyst, are made explicit and can be considered in the analysis of a specific aspect of the system.

- Time series modeling and prognosis of time series is a topic that is relevant especially in finance and economics. In this work we present for the first time an approach to generate auto-regressive models for multi-variate time series with genetic programming. In this context, a specialized fitness function that extends linear scaling also to time series prognosis is described. The proposed approach is applied to predict future values of a multi-variate financial time series. The resulting model approximates the long term of the time series accurately, however, the short term variations are not predicted correctly.
- In the second part the described methods are applied to three real-world data-analysis problems. The comprehensive symbolic regression approach produces interesting results for the chemical dataset and the economic dataset. In those two application scenarios no prior knowledge about variable interactions is available, thus the comprehensive modeling approach is reasonable to uncover implicit variable relations. In contrast, for the blast furnace process most of the underlying physical relations are already known a-priori and the untargeted modeling approach results in too many models. Additionally, most of the models are either not useful for the blast furnace operator or are only crude approximations of already known physical dependencies. Thus, we conclude that the comprehensive symbolic regression approach is useful when there is only little knowledge about the dataset, however, if many of the implicit relations are known already a more targeted modeling approach should be preferred.

The models produced by the comprehensive symbolic regression approach using the enhancements against bloat and overfitting are relatively accurate and comprehensible.

Needless to say this thesis does not claim to provide answers to all issues encountered in symbolic regression. Many new ideas and future research topics arise naturally from some results of this work. Four particular noteworthy topics are:

- Generalization of variable relevance metrics to other learning methods. The variable relevance measure based on permutation sampling can be used for any regression model, and as a matter of fact there have been approaches to apply it to neural networks and support vector machine models. Such approaches are interesting as the information gained through variable importance measures is often very insightful.
- Improving the multi-variate modeling approach through distance metrics for high-dimensional spaces. The experimental results show that multi-variate symbolic

regression models can be generated with genetic programming, however, the accuracies of the models are rather disappointing. It would be interesting to try if the model accuracy can be improved by using specialized distance metrics for high-dimensional spaces to calculate fitness of multi-variate models.

- Analysis of overfitting and model complexity. In this thesis various approaches to reduce overfitting are described, however, all approaches assume that overfitting can be reduced by reducing the solution length. Essentially program length and model complexity are treated as equal. This is slightly questionable, as semantically simple models can be represented as very long programs, and vice versa. Thus, it would be interesting to analyze overfitting in relation to model complexity and design methods to reduce overfitting by controlling model complexity instead of program length.
- Integration of a-priori knowledge for a targeted exploration of datasets. The comprehensive symbolic regression approach is not effective if a lot of a-priori knowledge about the modeled system is available, as for instance in the blast furnace application scenario. In such situations a more targeted modeling approach that takes a-priori knowledge into consideration is promising.

A. Additional Material

A.1. Model Accuracy Metrics

Throughout this work various different statistics are used to estimate the quality of models. The general quality of a prediction model is subjective and depends on the context, in which the model should be applied. The most relevant values for the model quality are its accuracy and its comprehensibility. The accuracy can be quantified more easily while the comprehensibility of a model is subjective. The accuracy of the model can be calculated from the output values \hat{y}_i of the given model and the actual target values y_i . A number of frequently used statistics for the accuracy of model output are defined in the following sections.

A.1.1. Regression Models

The quality of regression models $\hat{y}_i = g(x_i) = y_i + \epsilon$ can be determined by comparing the output values of the model \hat{y}_i with the actual target values y_i usually through the analysis of residuals $r_i = \hat{y}_i - y_i$. In the following the symbol p is used instead of \hat{y} for the predicted values of a model.

Mean Squared Error

The most well known accuracy metric for regression models is the mean squared error (MSE) function

$$\text{MSE}(\mathbf{p}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2, \quad (\text{A.1})$$

which is defined as the sum of the squared residuals of two vectors \mathbf{p}, \mathbf{y} with n elements over the number of elements n .

For many applications the MSE of the predicted values and the original target values is a good indicator for the loss incurred by incorrect predictions of the model. One problem of the MSE function is, that outliers can have strong influence on the result. Another problem is that MSE value depends on the location and scale of the original and estimated target variable values. Thus, the MSE is not invariant to linear transformations of the target variable values and MSE values of different input vectors \mathbf{y} are not comparable. The root mean squared error (RMSE)

$$\text{RMSE}(\mathbf{p}, \mathbf{y}) = \sqrt{\text{MSE}(\mathbf{p}, \mathbf{y})} \quad (\text{A.2})$$

is often reported in practice because it has the same dimension as the original values.

Normalized Mean Squared Error

Sometimes it is necessary to compare the accuracy of predictions for different target variables with different locations or scales. The MSE has the drawback, that it depends on the scale and the location of the target values, so the MSE of a prediction for one target variable cannot be readily compared to the MSE of a prediction for another target variable when the target variable values have not been normalized. The normalized mean squared error (NMSE) function

$$\begin{aligned}
 \text{NMSE}(\mathbf{p}, \mathbf{y}) &= \frac{1}{n} \sum_{i=1}^n \left(\frac{p_i - \bar{y}}{s_y} - \frac{y_i - \bar{y}}{s_y} \right)^2 \\
 &= \frac{1}{n} \frac{1}{s_y^2} \sum_{i=1}^n (p_i - y_i)^2 \\
 &= \frac{\text{MSE}(\mathbf{p}, \mathbf{y})}{s_y^2} \\
 s_y^2 &= \frac{1}{n-1} \sum_{i=1}^n (y_i - \bar{y})^2
 \end{aligned} \tag{A.3}$$

is one possibility to overcome the problem. In the NMSE function the input samples \mathbf{y} and \mathbf{p} are standardized using the mean \bar{y} and population variance s_y^2 of the original target variable. The NMSE value is a coefficient without dimension and the result for different input vectors can be compared easily, even with different locations and scales. The NMSE function is also relevant for the calculation of squared errors of the output of multi-variate regression models.

Multi-Variate Normalized Mean Squared Error

The extension of the normalized mean squared error to multi-variate inputs P and Y ,

$$\begin{aligned}
 P &= (p_{i,j})_{i=1\dots n, j=1\dots k}, \\
 Y &= (y_{i,j})_{i=1\dots n, j=1\dots k},
 \end{aligned}$$

with n values of dimension k , is the mean of the squared normalized Euclidean distances (A.5) of the actual values \mathbf{y}_i and the predicted values \mathbf{p}_i .

$$\text{MVNMSE}(P, Y) = \frac{1}{n} \sum_{i=1}^n d(\mathbf{p}_i, \mathbf{y}_i)^2 \tag{A.4}$$

$$\begin{aligned}
 d(\mathbf{p}_i, \mathbf{y}_i) &= \sqrt{\sum_{d=1}^k \left(\frac{p_{i,d} - \bar{y}_{\cdot,d}}{s_{y,d}} - \frac{y_{i,d} - \bar{y}_{\cdot,d}}{s_{y,d}} \right)^2} \\
 s_{y,d}^2 &= \frac{1}{n-1} \sum_{i=1}^n (y_{i,d} - \bar{y}_{\cdot,d})^2,
 \end{aligned} \tag{A.5}$$

where $s_{y,d}^2$ is the population variance of the d -th component of the original values of the target variable.

Mean Absolute Percentage Error

The mean absolute percentage error MAPE or relative error of a regression model is an intuitive and easily understandable accuracy metric. The MAPE of a vector of predicted values \mathbf{p} and a vector of original values \mathbf{y} is

$$\text{MAPE}(\mathbf{p}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n \left| \frac{p_i - y_i}{y_i} \right|. \quad (\text{A.6})$$

The MAPE function gives the average percental error of the predicted values \mathbf{p} relative to the original values \mathbf{y} . The result value is not defined if the original values contain elements equal to zero.

The MAPE functions is a generalization of the mean relative error (MRE) function

$$\text{MRE}(\mathbf{p}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n \frac{p_i - y_i}{y_i}, y_i > 0, \quad (\text{A.7})$$

which is not defined for target variables that can take negative or zero values. For target variable that only have strictly positive values, the MAPE and MRE metrics are equivalent.

Pearson's Product Moment Correlation Coefficient

Pearson's product moment correlation coefficient ρ (PPMC) is a measure for the correlation between two samples, in this case estimated values \mathbf{p} and the target values \mathbf{y} .

$$\begin{aligned} \rho(\mathbf{p}, \mathbf{y}) &= \frac{\text{Cov}(\mathbf{p}, \mathbf{y})}{\sqrt{\text{Var}(\mathbf{p})} \sqrt{\text{Var}(\mathbf{y})}} \\ \text{Cov}(\mathbf{p}, \mathbf{y}) &= \frac{1}{n-1} \sum_{i=1}^n (p_i - \bar{p})(y_i - \bar{y}) \\ \text{Var}(\mathbf{x}) &= \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2, \end{aligned} \quad (\text{A.8})$$

where $\text{Cov}(\mathbf{p}, \mathbf{y})$ is the sample covariance of vectors \mathbf{p} and \mathbf{y} , and $\text{Var}(\mathbf{x})$ is the sample variance of vector \mathbf{x} .

ρ can assume a value in the range of $[-1 \dots 1]$, where a value of -1 indicates perfectly indirectly correlated values, and a value of $+1$ indicates perfectly positive correlated values. A value of zero means there is no correlation of both variables. Note that $\rho = 0$ does not imply independence of the two samples; this is for instance discussed in [211] which introduces the distance correlation function. Usually the value of ρ^2 (R^2) is used as a metric for the accuracy of the model.

The PPMC is only applicable when the values of both samples stem from normally distributed random variables.

A.1.2. Time Series Models

All metrics for regression models can also be calculated for time series forecasts. For time series models, however, additional metrics for the accuracy of forecasts and predictions are frequently useful. Some of these accuracy metrics for time series forecasts are discussed in the following sections.

Mean Squared Error for Forecasts

The mean squared prediction error for time series forecasts \mathbf{p} with forecast horizon h for the original time series \mathbf{y} with n elements is

$$\text{MSPE}(\mathbf{p}, \mathbf{y}, h) = \frac{1}{n-h} \sum_{t=1}^{n-h} (p_{t+h} - y_{t+h})^2. \quad (\text{A.9})$$

This function can be generalized to multi-step forecasts P

$$P = (p_{t,j})_{t=1\dots n, j=1\dots h}, \quad (\text{A.10})$$

where one forecast \mathbf{p}_t is a vector of predictions $(p_{t,1}, p_{t,2}, \dots, p_{t,h})$ for horizons $j = 1 \dots h$ for the actual values \mathbf{y} . The MSPE for multi-step forecasts P is

$$\text{MSPE}(P, \mathbf{y}, h) = \frac{1}{n-h} \sum_{t=1}^{n-h} \frac{1}{h} \sum_{j=1}^h (p_{t,j} - y_{t+j})^2 w(j). \quad (\text{A.11})$$

The MSPE function for forecasts is simply the average of the squared errors of the predicted values $p_{t,j}$ relative to the actual values y_{t+j} over all forecasts t and horizons j , and thus very similar to the more familiar MSE function. Optionally forecasts for a given horizon j can be weighted using a weight function $w(j)$ to decrease the influence of predictions of large horizons relative to short term predictions.

Theil's U Statistic

Theil's inequality coefficient U^2 [213] is a statistic for the accuracy of forecasts of times. It relates the mean squared prediction error $\text{MSPE}(P, \mathbf{y}, h)$ of predictions P with the MSPE of a naive no-change forecast. Given a series of pairs of predicted changes P_i and observed changes A_i , Theil's inequality coefficient U is:

$$U^2 = \frac{\sum (P_i - A_i)^2}{\sum A_i^2} \quad (\text{A.12})$$

$\sum (P_i - A_i)^2$ in the numerator is the sum over all squared errors of pairs of predicted and realized changes. The denominator $\sum A_i^2$ is the sum of squared observed changes, which is equivalent to the squared error of a no-change model. Theil's inequality coefficient U can be interpreted as the root mean squared prediction error of the predicted changes over the root mean squared prediction error of naive no-change predictions. Usually the

model for the naive predictions is the no-change model $p_{t,h} = y_t$. If the underlying time series has a clear linear trend, the linear model $p_{t,h} = hdy_t + y_t$ should be used instead, where d is the linear trend of the series \mathbf{y} calculated as the mean of the percental changes of \mathbf{y} .

The inequality coefficient value can be easily interpreted. A perfect prediction has an inequality coefficient of zero, a prediction with an inequality coefficient between zero and one is better than a naive no-change prediction, and predictions with an inequality coefficient over one are worse than naive predictions.

In [213] relative predicted \hat{P}_t and relative realized changes \hat{A}_t are used for the analysis of economic forecasts because in this application the relative changes of economic variables over time are more interesting than the levels of the variables.

$$\hat{A}_t = \frac{a_t - a_{t-1}}{a_{t-1}} \quad (\text{A.13})$$

$$\hat{P}_t = \frac{p_t - a_{t-1}^*}{a_{t-1}^*} \quad (\text{A.14})$$

Where a_t is the actual level of the analyzed variable at time t , a_{t-1} is the actual level at the previous time step, p_t is the implied predicted level of the variable and a_{t-1}^* is the level of the variable at time $t-1$ as it was estimated at the time when the prediction was made. This distinction is made deliberately because, especially for economic forecasts, the value of a variable at a given time might be adjusted over time as more data become available.

To overcome the problem that relative changes are asymmetric (a 5 percent increase followed by a 5 percent decrease does not accumulate to the original value) Theil worked with the logarithms of $(1 + \hat{P}_t)$ and $(1 + \hat{A}_t)$:

$$A_t = \log(1 + \hat{A}_t) \quad (\text{A.15})$$

$$P_t = \log(1 + \hat{P}_t) \quad (\text{A.16})$$

This approach cannot be used when the variables can take negative or zero values. In such cases alternative definitions of realized or observed changes A_i and predicted changes P_i must be used.

Given n pairs of one step forecasts \hat{y}_t from any time series prognosis model and observed values of the time series y_t , the $(n-1)$ pairs of predicted changes P_i and observed changes A_i can be defined as:

$$P_i = \hat{y}_t - y_{t-1} \quad (\text{A.17})$$

$$A_i = y_t - y_{t-1} \quad (\text{A.18})$$

The definition of the inequality coefficient can be extended to multi-step forecasts up to a finite positive horizon h . In this case a forecast at time t is a combination h predictions $\hat{y}_{t+1}, \hat{y}_{t+2}, \dots, \hat{y}_{t+h}$ which can be paired with the actually observed values

$y_{t+1}, y_{t+2}, \dots, y_{t+h}$ for the calculation of the inequality coefficient of the predictions. Given a multi-step forecast at time t for horizons $i \in [1..h]$ the set of pairs of predicted changes $P_{t,h}$ and realized changes $A_{t,h}$ for this forecast is:

$$P_{t,i} = \hat{y}_{t+i} - y_t \quad (\text{A.19})$$

$$A_{t,i} = y_{t+i} - y_t \quad (\text{A.20})$$

Given n observed values of the time series y_t we can calculate the inequality coefficient for $(n - h)$ multi-step forecasts as:

$$U^2 = \frac{\sum_{t=1}^{n-h} \sum_{i=1}^h (P_{t,i} - A_{t,i})^2}{\sum_{t=1}^{n-h} \sum_{i=1}^h A_{t,i}^2} \quad (\text{A.21})$$

Directional Symmetry

The directional symmetry (DS) of time series forecast is a statistic for the accuracy of predictions of directional changes. The directional symmetry is the percentage of directional changes of the target variable that are predicted correctly.

$$\text{DS} = 100 \frac{1}{n-h} \sum_{t=0}^{n-h} \sum_{i=1}^h d_{t,i} \quad (\text{A.22})$$

$$d_{t,i} = \begin{cases} 1, & \text{if } P_{t,i} A_{t,i} \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (\text{A.23})$$

$$P_{t,i} = \hat{y}_{t+i} - y_t \quad (\text{A.24})$$

$$A_{t,i} = y_{t+i} - y_t \quad (\text{A.25})$$

Weighted Directional Symmetry

The weighted directional symmetry includes the size of the change into the metric. Because of its discrete definition the directional symmetry metric does not give a good estimation of the quality of the model. For instance, if the model can forecast small directional changes correctly, but cannot forecast large directional changes, then the quality of the model is actually worse than the directional symmetry indicates. For this reason the weighted directional symmetry includes the size of the change.

$$\text{WDS} = 100 \frac{1}{n-h} \sum_{t=0}^{n-h} \sum_{i=1}^h d_{t,i} |y_{t+i} - \hat{y}_{t+i}| \quad (\text{A.26})$$

$$d_{t,i} = \begin{cases} 1, & \text{if } P_{t,i} A_{t,i} \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (\text{A.27})$$

$$P_{t,i} = \hat{y}_{t+i} - y_t \quad (\text{A.28})$$

$$A_{t,i} = y_{t+i} - y_t \quad (\text{A.29})$$

The weighted directional symmetry of models which can predict large directional changes better might have a better weighted directional symmetry, than models that predict more of the total directional changes correctly but fail at predicting large directional changes.

A.2. Numerical Approximation of Derivatives

In some situations it is necessary to calculate derivatives of input variables before, as a preprocessing step for modeling. This is common for time series modeling where $\frac{dx}{dt}$ is either the target variable or used as an input variable. Because the generating function of the variable values is generally not known in such situations, the derivative can only be approximated numerically using the original data of variable x . The three point formula A.30 or the slightly more accurate five-point approximation A.31 are central approximations and use forward and backward data of the series. Strictly forward A.32 or backward A.33 approximations are also useful for instance to calculate derivatives at the endpoints of a data series. Also see [167] for more details about numeric approximations of derivatives and integrals.

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \quad (\text{A.30})$$

$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} \quad (\text{A.31})$$

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} \quad (\text{A.32})$$

$$f'(x) = \frac{+3f(x) - 4f(x-h) + f(x-2h)}{2h} \quad (\text{A.33})$$

It has to be noted that numeric approximations of derivatives are problematic when the original data series contains noisy values, because the noise is amplified when calculating the derivative. Often it is necessary to smooth the original data series before its derivative can be calculated, for instance using a Savitzky-Golay filter [175, 200, 167]. The drawback of smoothing is that relevant information, that is necessary for modeling, might be lost. A recent contribution discussing numeric derivation of noisy data is [33].

A.3. Datasets Used in Experiments

In this section the origin and details about the datasets, used in experiments in this work, are given. All data sets can be downloaded from <http://dev.heuristiclab.com/trac/hl/core/wiki/AdditionalMaterial>, either in comma-separated values format or as HeuristicLab problem files.

A.3.1. Artificial benchmark datasets

Friedman I

This dataset is described in [71] where it is used to benchmark the multi-variate adaptive regression splines (MARS) algorithm. The signal-to-noise ratio in this dataset is rather low, so it is difficult to rediscover the generating function, especially the terms below the noise level (x_4 and x_5).

x_1, \dots, x_{10} are sampled uniformly from the unit hypercube.

$$\begin{aligned} f(\mathbf{x}) &= 0.1e^{4x_1} + \frac{4}{1 + e^{-20(x_2-0.5)}} + 3x_3 + 2x_4 + x_5 \\ y_i &= f(x_i) + \epsilon_i, 1 \leq i \leq N \end{aligned} \quad (\text{A.34})$$

ϵ_i is generated from the standard normal distribution.

This dataset is used to analyze different variable relevance metrics for GP in Chapter 6, and to study bloat and anti-bloat measures in Chapter 4.

Friedman II

This dataset is also described in [71] and used to benchmark the MARS algorithm. The signal-to-noise ratio is high, so it is easier to rediscover the generating function than for the Friedman-I dataset.

x_1, \dots, x_{10} are sampled uniformly from the unit hypercube.

$$f(\mathbf{x}) = 10 \sin(\pi x_1 x_2) + 20(x_3 - \frac{1}{2})^2 + 10x_4 + 5x_5 + \sigma(0, 1) \quad (\text{A.35})$$

This dataset is used to analyze different variable relevance metrics for GP in Chapter 6, and to study bloat and anti-bloat measures in Chapter 4.

Breiman I

This dataset is described in [28] where it is used to benchmark the classification and regression trees (CART) algorithm. The signal-to-noise ratio is rather low and additionally it contains a crisp conditional which makes it rather difficult to rediscover the generating function with a symbolic regression approach.

x_1, \dots, x_{10} are randomly sampled attributes following the probability distributions:

$$\begin{aligned} P(x_1 = -1) &= P(x_1 = 1) = \frac{1}{2} \\ P(x_m = -1) &= P(x_m = 0) = P(x_m = 1) = \frac{1}{3}, m = 2, \dots, 10 \\ f(\mathbf{x}) &= \begin{cases} x_1 = 1, & 3 + 3x_2 + 2x_3 + x_4 \\ \text{otherwise} & -3 + 3x_5 + 2x_6 + x_7 \end{cases} \\ y &= f(\mathbf{x}) + \epsilon_i \\ \epsilon &\sim N(0, 2) \end{aligned} \quad (\text{A.36})$$

This dataset is used to analyze different variable relevance metrics for GP in Chapter 6, and to study bloat and anti-bloat measures in Chapter 4.

A.3.2. Real World Datasets

Chemical-I

This dataset has been prepared and published by Arthur Kordon, research leader at the Dow Chemical company for the EvoCompetitions side event of the EvoStar conference 2010. The dataset stems from a real industrial process and contains 747+319 observations of 58 variables. The values of the target variable are noisy lab measurements of the chemical composition of the product which are expensive to measure. The remaining 57 variables are material flows, pressures, temperatures collected from the process which can be measured easily.

This dataset is used in Chapter 4 to study bloat and anti-bloat measures, and in Chapter 5 to analyze overfitting and anti-overfitting methods for GP. In Chapter 8 this dataset is studied in full detail.

Chemical-II

This dataset is the “Tower” dataset which has also been prepared and published by Arthur Kordon, Dow Chemical. The dataset contains 4999 observations of 26 variables of a chemical process and can be downloaded from <http://vanillamodeling.com/realproblems.html>. The following description of the Tower dataset also stems from the same URL.

The observations in the dataset stem from a chemical process and include temperatures, flows, and pressures. The target variable is the propylene concentration and is a gas chromatography measurement at the top of a distillation tower. The propylene concentration is measured in regular intervals of 15 minutes. The actual sampling rate of the input variables is one minute, but 15 minutes averages of the inputs are used to synchronize with the measurements of the target variable. The range of the measured propylene concentration is very broad and covers most of the expected operating conditions in the distillation tower.

The dataset is used in this work to study different variable relevance metrics for GP in Chapter 6.

Blast Furnace Dataset

This dataset contains measurements of a blast furnace for the production of liquid iron at the voestalpine plant in Linz. The dataset contains hourly measurements over a time span of several years and is treated as a time series. The observations stem from a tightly controlled blast furnace process. The data set contains sensitive information and thus is unfortunately not freely available.

This dataset is studied in full detail in Chapter 8.

Financial-I

The dataset has been prepared by Ricardo de A. Araújo and Glaucio G. de M. Melo for a financial time series prediction competition which was held as a side event of the EvoStar conference 2010 [43]. Only little information about the origin of the dataset has been published, except that the dataset stems from real financial time series from the Brazilian stock exchange.

The original dataset contains 200 observations of ten financial time series. The original competition objective was to predict the next 10 data-points for each of the 10 time series. Unfortunately the 10 data points, which were originally held back by the organizers to evaluate the entries, have never been published.

The dataset was used to demonstrate GP-based multi-variate time series prognosis in Chapter 7

Macro-Economic

This dataset has been prepared and provided to us by Dr. Stefan Fink. The dataset contains 331 monthly observations of 33 macro economic variables mainly from the US in the time span from January 1980 until July 2007.

The dataset is studied in full detail in Chapter 8

Housing

This dataset is the Boston housing dataset from the UCI machine learning repository [69] and can be downloaded from <http://archive.ics.uci.edu/ml/datasets/Housing>. The original data source is the StatLib library, which is maintained at Carnegie Mellon University. The dataset contains 506 observations of 14 variables concerning the housing values in the suburbs around Boston.

The dataset is used in this thesis to study overfitting and anti-overfitting measures for GP in Chapter 5.

List of Figures

2.1. Underfitting and overfitting	17
3.1. Example for a simple symbolic regression model and the equivalent expression in mathematical notation.	30
4.1. Best solution quality and bloat comparison of OSGP with static limits and OSGP with dynamic depth limits (average values over thirty independent runs, x-axis is generation index).	43
4.2. Best solution quality and bloat comparison of SGP-static, SGP-DynOpEq, SGP-DDL, and SGP-CPP (average values over thirty runs, x-axis shows generation index).	48
4.3. Best quality and bloat of SGP and OSGP without size limits (average values over thirty independent GP runs, x-axis shows number of generations).	53
4.4. Best quality and bloat of SGP and OSGP with static size limits (average values over thirty independent GP runs, x-axis shows number of generations).	55
4.5. Model simplification as implemented in HeuristicLab	59
4.6. Original symbolic regression solution as produced by GP for the Housing problem	59
4.7. Automatically simplified solution	60
4.8. Manually pruned solution and mathematical representation with all constants removed.	60
5.1. Trajectories of training and test fitness and $\rho(f_{\text{training}}(g), f_{\text{validation}}(g))$ over 100 generations in an exemplary run of SGP for the housing dataset. The scatter plots in the lower half show the training vs. validation fitness of all models in the population at four different generations.	69
5.2. Overfitting behavior of SGP and OSGP without size limits and without overfitting control.	70
5.3. Overfitting behavior of SGP-static, OSGP-static, and SGP-CPP without overfitting control.	71
5.4. Overfitting behavior of SGP, SGP adaptive covariant parsimony pressure, and SGP with overfitting-triggered pruning.	72
5.5. Overfitting behavior of OSGP-static, OSGP with overfitting-triggered pruning, and OSGP with overfitting-triggered pruning based on the validation set.	73

6.1.	Artificial function $h_1(x, y)$ and its response surface	83
6.2.	Squared difference functions $\Delta_{h_1,x}(x, y)$ and $\Delta_{h_1,y}(x, y)$	84
6.3.	Artificial function $h_2(x, y)$ and its response surface	85
6.4.	Squared difference functions $\Delta_{h_2,x}(x, y)$ and $\Delta_{h_2,y}(x, y)$	86
6.5.	Trajectories of relative variable frequencies over a single GP run for a benchmark dataset.	88
6.6.	Kernel density estimation of frequency-based variable impacts for the Breiman-I dataset.	91
6.7.	Scatter plot of actual variable impact and variable relevance values. The Rel _{MC} metric is very accurate with a correlation coefficient $R^2 = 0.97$. The Rel _{mean} metric has a correlation coefficient of $R^2 = 0.92$	93
6.8.	Box-plot of R^2 on the test set of the model output and original values for the Chemical-II dataset.	98
6.9.	Identified variable relations for the Chemical-II dataset.	100
7.1.	A multi-variate symbolic regression model has a separate branch for each component of the result vector below the result producing branch (RPB).	105
7.2.	Multi-variate symbolic regression model with result producing branch (RPB) and two automatically defined functions (ADF0, ADF1)	106
7.3.	Model of evaluation for the 5-step prognosis of financial time series.	114
7.4.	Box-plot of squared correlation coefficient (R^2) over 80 models for each component of the multi-variate financial time series.	114
7.5.	Boxplot of Theil's U over 80 models for each component of the multi- variate financial time series. The horizontal line indicates the Theil's U of the naive prognosis.	115
7.6.	Original values and predicted values (horizon=10) of the model (7.11) for the financial time series dataset.	117
7.7.	Actual continuation and predicted continuation (horizon=1..10) of the model (7.11) for the test partition (rows 190–200) of the financial time series dataset.	118
8.1.	Schematic diagram of the blast furnace and input and output materials. At the top ferrous oxides and coke are charged in alternating layers. Near the bottom the hot blast and reducing agents are injected through the tuyeres. Hot metal and slag are tapped in regular intervals from the hearth of the blast furnace. Blast furnace gas is collected at the top of the blast furnace.	120
8.2.	Box-plot of R^2 value on the test set of models for the blast furnace dataset.	123
8.3.	Relationships of blast furnace variables identified with unguided GP-based data mining. Arrows indicate a dependency in the aspect of data modeling and do not necessarily match physical or chemical causations.	125
8.4.	Scatter plot of the original values of Pressure _{HB} and the output of model 8.1.	126

8.5. Box-plot of R^2 values on the test set of models for the macro-economic dataset.	135
8.6. Relationships of macro-economic variables identified with comprehensive symbolic regression and frequency-based variable relevance metrics. This figure has been plotted using GraphViz.	136
8.7. Line chart of the actual value of the <i>US Help wanted index</i> and the estimated values produced by the model (Equation 8.16). Test set starts at index 300.	139
8.8. Line chart of the actual value of the <i>US CPI inflation</i> and the estimated values produced by the model (Equation 8.19).	141
8.9. Box-plot of R^2 value on the test set of models for the chemical dataset. .	146
8.10. Relationships of variables in the Chemical-I dataset identified with unguided GP-based data mining	147

List of Tables

4.1. Parameter settings for the OSGP-DDL experiments.	41
4.2. Parameter settings for SGP-DynOpEq, SGP-DDL, and SGP-CPP experiments.	47
4.3. Genetic programming parameter settings for the analysis of bloat in SGP and OSGP.	51
4.4. Summary of results of bloat experiments for the Breiman-I problem (averages over 30 runs, confidence intervals for $\alpha = 0.05$).	56
4.5. Summary of results of bloat experiments for the Friedman-I problem (averages over 30 runs, confidence intervals for $\alpha = 0.05$).	56
4.6. Summary of results of bloat experiments for the Chemical-I problem (averages over 30 runs, confidence intervals for $\alpha = 0.05$).	57
5.1. Genetic programming parameters for the experiments.	67
5.2. Summary of best training solution results of overfitting experiments for the Chemical-I problem (confidence intervals for $\alpha = 0.05$).	74
5.3. Summary of best training solution results of overfitting experiments for the Housing problem (confidence intervals for $\alpha = 0.05$).	75
6.1. Genetic programming parameter settings for the validation of variable relevance metrics.	90
6.2. Actual variable impacts Impact_{fun} for the Breiman-I function and variable relevance results over ten independent GP runs.	91
6.3. Actual variable impacts Impact_{fun} for the Friedman-I function and variable relevance results over ten independent GP runs.	92
6.4. Actual variable impacts Impact_{fun} for the Friedman-II function and variable relevance results over ten independent GP runs.	93
6.5. Variable relevance results for the Dow Chemical tower dataset over ten independent GP runs.	94
6.6. Groups of variables with strong pairwise correlations in the Chemical-II dataset.	97
6.7. Genetic programming parameters for the Chemical-II dataset.	98
6.8. Accuracy of selected and simplified models for the Chemical-II dataset on the test set.	101
7.1. Genetic programming parameter settings for the financial time series prognosis experiments.	113

8.1. Genetic programming parameters for the blast furnace dataset.	122
8.2. Variables included in the blast furnace dataset.	122
8.3. Overview of selected models for the blast furnace process.	125
8.4. Scaling parameters α and β for the model for $\text{Pressure}_{\text{HB}}$ shown in Equation 8.2 for different partitions of the blast furnace data set.	127
8.5. List of economic variables considered for the identification of macro-economic relations	132
8.6. Genetic programming parameters.	134
8.7. Overview model accuracy for the simplified macro-economic models identified by GP.	137
8.8. Clusters of strongly correlated variables in the chemical dataset. The variable used as representative for the cluster in the modeling process is indicated in bold font.	144
8.9. Genetic programming parameters for the symbolic regression experiments with the chemical dataset.	145
8.10. Overview of models for the chemical dataset.	148

List of Algorithms

- 1. Dynamic depth limits for offspring selection 40
- 2. Filtering of offspring in the formulation of dynamic depth limits and operator equalization 42
- 3. Acceptance criterion of dynamic operator equalization 45
- 4. Acceptance criterion of dynamic operator equalization with offspring selection 46
- 5. Greedy iterated tournament pruning 50
- 6. Algorithm for overfitting detection. 67
- 7. Linear scaling and fitness evaluation of time series prognosis models 112

Bibliography

- [1] Katharine G. Abraham and Michael Wachter. Help-wanted advertising, job vacancies, and unemployment. *Brookings Papers on Economic Activity*, pages 207–248, 1987.
- [2] Michael Affenzeller and Stefan Wagner. Offspring selection: A new self-adaptive selection scheme for genetic algorithms. In B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, editors, *Adaptive and Natural Computing Algorithms*, Springer Computer Series, pages 218–221. Springer, 2005.
- [3] Michael Affenzeller, Stephan Winkler, Stefan Wagner, and Andreas Beham. *Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications*, volume 6 of *Numerical Insights*. CRC Press, Chapman & Hall, 2009.
- [4] Michael Affenzeller, Stephan M. Winkler, and Stefan Wagner. Effective allele preservation by offspring selection: An empirical study for the TSP. *International Journal of Simulation and Process Modelling*, 6(1):29–39, 2010.
- [5] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. On the surprising behavior of distance metrics in high dimensional space. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory – ICDT 2001*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer Berlin / Heidelberg, 2001.
- [6] Sameer H. Al-Sakran, John R. Koza, and Lee W. Jones. Automated re-invention of a previously patented optical lens system using genetic programming. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 25–37, Lausanne, Switzerland, 30 March - 1 April 2005. Springer.
- [7] Enrique Alba. *Parallel metaheuristics: a new class of algorithms*. Wiley-Interscience, 2005.
- [8] Eva Alfaro-Cid, Anna Esparcia-Alcázar, Ken Sharman, Francisco Fernández de Vega, and J. J. Merelo. Prune and plant: A new bloat control method for genetic programming. In *Proceedings of the 2008 8th International Conference on Hybrid Intelligent Systems*, pages 31–35, Washington, DC, USA, 2008. IEEE Computer Society.
- [9] Eva Alfaro-Cid, Juan J. Merelo, Francisco Fernández de Vega, Anna I. Esparcia-Alcázar, and Ken Sharman. Bloat control operators and diversity in genetic programming: A comparative study. *Evolutionary Computation*, 18(2):305–332, 2010.

- [10] Lee Altenberg. Emergent phenomena in genetic programming. In Anthony V. Sebald and Lawrence J. Fogel, editors, *Evolutionary Programming — Proceedings of the Third Annual Conference*, pages 233–241, San Diego, CA, USA, 24–26 February 1994. World Scientific Publishing.
- [11] Dietmar Andahazy, Gerhard Löffler, Franz Winter, Christoph Feilmayr, and Thomas Bürgler. Theoretical analysis on the injection of H_2 , CO , CH_4 rich gases into the blast furnace. *ISIJ International*, 45(2):166–174, 2005.
- [12] Dietmar Andahazy, Sabine Slaby, Gerhard Löffler, Franz Winter, Christoph Feilmayr, and Thomas Bürgler. Governing processes of gas and oil injection into the blast furnace. *ISIJ International*, 46(4):496–502, 2006.
- [13] David Andre and John R. Koza. Parallel genetic programming: A scalable implementation using the transputer network architecture. In Peter J. Angeline and K. E. Kinneer, Jr., editors, *Advances in Genetic Programming 2*, chapter 16, pages 317–338. MIT Press, Cambridge, MA, USA, 1996.
- [14] David Andre, Forrest H Bennett III, and John R. Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 3–11, Stanford University, CA, USA, 28–31 July 1996. MIT Press.
- [15] Peter J. Angeline. An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 21–29, Stanford University, CA, USA, 28–31 July 1996. MIT Press.
- [16] Peter J. Angeline. A historical perspective on the evolution of executable structures. *Fundamenta Informaticae*, 35(1–4):179–195, August 1998. ISSN 0169-2968.
- [17] Peter John Angeline. *Evolutionary Algorithms and Emergent Intelligence*. PhD thesis, Ohio State University, 1993.
- [18] Francesco Archetti, Ilaria Giordani, and Leonardo Vanneschi. Genetic programming for anticancer therapeutic response prediction using the NCI-60 dataset. *Computers & Operations Research*, 37(8):1395–1405, 2010. Operations Research and Data Mining in Biological Systems.
- [19] V. Arkov, C. Evans, P. J. Fleming, D. C. Hill, J. P. Norton, I. Pratt, D. Rees, and K. Rodriguez-Vazquez. System identification strategies applied to aircraft gas turbine engines. *Annual Reviews in Control*, 24(1):67–81, 2000.

- [20] W. Banzhaf and W. B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91, March 2002. ISSN 1389-2576.
- [21] Wolfgang Banzhaf. Genetic programming for pedestrians. MERL Technical Report 93-03, Mitsubishi Electric Research Labs, Cambridge, MA, USA, 1993.
- [22] Stefan Bleuler, Martin Brack, Lothar Thiele, and Eckart Zitzler. Multiobjective genetic programming: Reducing bloat using SPEA2. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 536–543, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.
- [23] George Box and Gwilym Jenkins. *Time series analysis: forecasting and control*. Holden-Day, Oakland, California, 1976.
- [24] Leo Breiman. Out-of-bag estimation. Technical report, Statistics Department, University of California, 1996.
- [25] Leo Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.
- [26] Leo Breiman. Random forest. *Machine Learning*, 45, 2001.
- [27] Leo Breiman. Statistical modeling: The two cultures. *Statistical Science*, 16(3): 199–231, 2001.
- [28] Leo Breiman, Jerome H. Friedman, Charles J. Stone, and R.A. Olson. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.
- [29] Peter J. Brockwell and Richard A. Davis. *Time Series and Forecasting Methods*. Springer, New York, second edition, 1991.
- [30] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer, New York, 1996.
- [31] Matthew Butler and Vlado Keselj. Optimizing a pseudo financial factor model with support vector machines and genetic programming. In Yong Gao and Nathalie Japkowicz, editors, *22nd Canadian Conference on Artificial Intelligence, Canadian AI 2009*, volume 5549 of *Lecture Notes in Computer Science*, pages 191–194, Kelowna, Canada, May 25-27 2009. Springer.
- [32] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [33] Rick Chartrand. Numerical differentiation of noisy, nonsmooth data. Submitted, 2010.

- [34] Sin Man Cheang, Kwong Sak Leung, and Kin Hong Lee. Genetic parallel programming: Design and implementation. *Evolutionary Computation*, 14(2):129–156, Summer 2006.
- [35] Jian Chen. A predictive system for blast furnaces by integrating a neural network with qualitative analysis. *Engineering Applications of Artificial Intelligence*, 14(1):77–85, 2001.
- [36] Shu-Heng Chen, Hung-Shuo Wang, and Byoung-Tak Zhang. Forecasting high-frequency financial time series with evolutionary neural trees: The case of hang-seng stock index. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Artificial Intelligence, IC-AI '99*, volume 2, pages 437–443, Las Vegas, Nevada, USA, 28 June-1 July 1999. CSREA Press.
- [37] Darren M. Chitty. A data parallel approach to genetic programming using programmable graphics hardware. In Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1566–1573, London, 7-11 July 2007. ACM Press.
- [38] William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74:829–836, 1979.
- [39] William S. Cleveland and Susan J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83:596–610, 1988.
- [40] Malcolm S. Cohen and Robert M. Solow. The behavior of help-wanted advertising. *The Review of Economics and Statistics*, 49(1):108–110, Feb. 1967.
- [41] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. ISSN 0885-6125. 10.1007/BF00994018.
- [42] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, January 1967.
- [43] Ricardo de A. Araújo and Glaucio G. de M. Melo. Financial time series competition in evo*. online: <http://www.gm2.com.br/ftspc/>, April 2010.
- [44] Edwin D. de Jong and Jordan B. Pollack. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*, 4(3):211–233, September 2003.

- [45] Edwin D. de Jong, Richard A. Watson, and Jordan B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 11–18, San Francisco, California, USA, 7–11 July 2001. Morgan Kaufmann.
- [46] Francisco Fernandez de Vega, Juan M. Sanchez, Marco Tomassini, and Juan A. Gomez. A parallel genetic programming tool based on PVM. In J. Dongarra, E. Luque, and T. Margalef, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proceedings of the 6th European PVM/MPI Users' Group Meeting*, volume 1697 of *Lecture Notes in Computer Science*, pages 241–248, Barcelona, Spain, September 1999. Springer-Verlag.
- [47] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 1 edition, 2001.
- [48] Kalyanmoy Deb, Amrit Pratap, Samir Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6, April 2002.
- [49] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum-likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.
- [50] Michael A. H. Dempster and Chris M. Jones. A real-time adaptive trading system using genetic programming. *Quantitative Finance*, 1:397–413, 2000.
- [51] Michael A. H. Dempster, Tom W. Payne, Yazann Romahi, and G. W. P. Thompson. Computational learning techniques for intraday FX trading using popular technical indicators. *IEEE Transactions on Neural Networks*, 12(4):744–754, July 2001.
- [52] Stephen Dignum and Riccardo Poli. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1588–1595, London, 7–11 July 2007. ACM Press.
- [53] Stephen Dignum and Riccardo Poli. Crossover, sampling, bloat and the harmful effects of size limits. In *Proceedings of the 11th European conference on Genetic programming*, EuroGP'08, pages 158–169, Berlin, Heidelberg, 2008. Springer-Verlag.

- [54] Stephen Dignum and Riccardo Poli. Operator equalisation and bloat free GP. In Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, Anna Isabel Esparcia Alcazar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino, editors, *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 110–121, Naples, 26–28 March 2008. Springer.
- [55] Stephen Dignum and Riccardo Poli. Sub-tree swapping crossover and arity histogram distributions. In Anna Isabel Esparcia-Alcazar, Aniko Ekart, Sara Silva, Stephen Dignum, and A. Sima Uyar, editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 38–49, Istanbul, 7–9 April 2010. Springer.
- [56] Pedro Domingos. The role of occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3:409–425, 1999. ISSN 1384-5810.
- [57] James Durbin and Siem Jan Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, 2001.
- [58] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003. ISBN 3-540-40184-9.
- [59] Agoston E. Eiben and Mark Jelasity. A critical note on experimental research methodology in EC. In *In Congress on Evolutionary Computation (CEC'02)*, pages 582–587. IEEE Press, 2002.
- [60] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 2010.
- [61] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazin*, 17(3):37–54, 1996.
- [62] Christoph Feilmayr. personal communication, 2010.
- [63] Barry Feldman. Relative importance and value. Manuscript (latest version), downloadable at <http://www.prismanalytics.com/docs/RelativeImportance.pdf>, 2005.
- [64] David Firth. Relative importance of explanatory variables. Conference on Statistical Issues in the Social Sciences, Stockholm, October 1998. Online at <http://www.nuff.ox.ac.uk/sociology/alcd/relimp.pdf>., October 1998.
- [65] Evelyn Fix and Joseph L. Hodges. Discriminatory analysis, nonparametric discrimination: consistency properties. Technical Report 4, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.
- [66] David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, 1995.

- [67] Lawrence J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.
- [68] Stephanie Forrest, ThanhVu Nguyen, Westley Weimer, and Claire Le Goues. A genetic programming approach to automated software repair. In Guenther Raidl, Franz Rothlauf, Giovanni Squillero, Rolf Drechsler, Thomas Stuetzle, Mauro Birattari, Clare Bates Congdon, Martin Middendorf, Christian Blum, Carlos Cotta, Peter Bosman, Joern Grahl, Joshua Knowles, David Corne, Hans-Georg Beyer, Ken Stanley, Julian F. Miller, Jano van Hemert, Tom Lenaerts, Marc Ebner, Jaume Bacardit, Michael O'Neill, Massimiliano Di Penta, Benjamin Doerr, Thomas Jansen, Riccardo Poli, and Enrique Alba, editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 947–954, Montreal, Québec, Canada, 8–12 July 2009. ACM. ISBN 978-1-60558-325-9.
- [69] A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL <http://archive.ics.uci.edu/ml>.
- [70] Alex Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, August 2002. (264 pages).
- [71] Jerome H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–141, 1991.
- [72] Jerome H. Friedman. Data mining and statistics: What's the connection? online, November 1997. <http://www-stat.stanford.edu/~jhf/ftp/dm-stat.pdf>.
- [73] Christian Gagne, Marc Schoenauer, Marc Parizeau, and Marco Tomassini. Genetic programming, validation sets, and parsimony pressure. In *Genetic Programming, 9th European Conference, EuroGP2006*, volume 3905 of *Lecture Notes in Computer Science*, pages 109–120, Berlin, Heidelberg, New York, 2006. Springer.
- [74] G. Gardner, Andrew C. Harvey, and Garry D. A. Phillips. Algorithm AS154. an algorithm for exact maximum likelihood estimation of autoregressive-moving average models by means of Kalman filtering. *Applied Statistics*, 29:311–322, 1980.
- [75] M. Giacobini, M. Tomassini, and L. Vanneschi. Limiting the number fitness cases in genetic programming using statistics. In J.J. Merelo-Guervos, P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacanas, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 371–380. Springer, 2002.
- [76] Phillip Good. *Permutation, parametric, and bootstrap tests of hypotheses*. Springer-Verlag, New York, 2005.
- [77] Ulrike Grömping. Estimators of relative importance in linear regression based on variance decomposition. *The American Statistician*, 61(2):132–138, May 2007.

- [78] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.
- [79] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. The MIT Press, 2001.
- [80] Simon Harding and Wolfgang Banzhaf. Fast genetic programming on GPUs. In Marc Ebner, Michael O’Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 90–101, Valencia, Spain, 11-13 April 2007. Springer.
- [81] Simon Harding, Julian F. Miller, and Wolfgang Banzhaf. Developments in cartesian genetic programming: self-modifying CGP. *Genetic Programming and Evolvable Machines*, 11(3/4):397–439, September 2010. ISSN 1389-2576. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.
- [82] Andrew C. Harvey. *Time Series Models*. Harvester Wheatsheaf, 2nd edition, 1993.
- [83] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*. Springer, 2009. Second Edition.
- [84] Christopher J. Hillar and Friedrich T. Sommer. On the article ”Distilling free-form natural laws from experimental data”. Retrieved Oct. 2010, 2009. URL <http://www.msri.org/people/members/chillar/files/hs09b.pdf>.
- [85] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, July 2006.
- [86] John Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.
- [87] Jeff W. Johnson and James M. Lebetron. History and use of relative importance indices in organizational research. *Organizational Research Methods*, 4:238–257, July 2004.
- [88] Mark Johnston, Thomas Liddle, and Mengjie Zhang. A relaxed approach to simplification in genetic programming. In Anna Esparcia-Alcázar, Anikó Ekárt, Sara Silva, Stephen Dignum, and A. Uyar, editors, *Genetic Programming*, volume 6021 of *Lecture Notes in Computer Science*, pages 110–121. Springer Berlin / Heidelberg, 2010.
- [89] I.T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, NY, 2nd edition, 2002.
- [90] Richard H. Jones. Maximum likelihood fitting of ARMA models to time series with missing observations. *Technometrics*, 20:389–395, 1980.

- [91] Hugues Juille and Jordan B. Pollack. Massively parallel genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 17, pages 339–358. MIT Press, Cambridge, MA, USA, 1996.
- [92] Mak Kaboudan. A measure of time series predictability using genetic programming applied to stock returns. *Journal of Forecasting*, 18:345–357, 1999.
- [93] Mak Kaboudan. Extended daily exchange rates forecasts using wavelet temporal resolutions. *New Mathematics and Natural Computing*, 1:79–107, 2005.
- [94] Maarten Keijzer. *Scientific Discovery using Genetic Programming*. PhD thesis, Lyngby, Denmark, May 2002.
- [95] Maarten Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, September 2004.
- [96] Kenneth E. Kinnear, Jr. Alternatives in automatic function definition: A comparison of performance. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 6, pages 119–141. MIT Press, 1994.
- [97] David Kinzett, Mark Johnston, and Mengjie Zhang. Numerical simplification for bloat control and analysis of building blocks in genetic programming. *Evolutionary Intelligence*, 2:151–168, 2009. ISSN 1864-5909.
- [98] Michael Kommenda, Gabriel K. Kronberger, Michael Affenzeller, Stephan M. Winkler, Christoph Feilmayr, and Stefan Wagner. Symbolic regression with sampling. In *Proc. of the 22nd European Modeling and Simulation Symposium EMSS 2010*, pages 13–18, Fes, Marokko, 2010.
- [99] Mark Kotanchek, Guido Smits, and Ekaterina Vladislavleva. Trustable symbolic regression models: Using ensembles, interval arithmetic and pareto fronts to develop robus and trust-aware models. In Rick Riolo, Terence Soule, and Bill Worzel, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, pages 203–222. Springer, 2008.
- [100] John R. Koza. A genetic approach to econometric modeling. In *Sixth World Congress of the Econometric Society*, Barcelona, Spain, 1990.
- [101] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [102] John R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3–4):251–284, 2010.
- [103] John R. Koza, Forrest H Bennett III, David Andre, and Martin A. Keane. Four problems for which a computer program evolved by genetic programming is competitive with human performance. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, volume 1, pages 1–10. IEEE Press, 1996.

- [104] John R. Koza, Forrest H Bennett III, and Oscar Stiffelman. Genetic programming as a Darwinian invention machine. In Riccardo Poli, Peter Nordin, William B. Langdon, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'99*, volume 1598 of *LNCS*, pages 93–108, Goteborg, Sweden, 26-27 May 1999. Springer-Verlag.
- [105] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.
- [106] Gabriel Kronberger, Christoph Feilmayr, Michael Kommenda, Stephan Winkler, Michael Affenzeller, and Thomas Burgler. System identification of blast furnace processes with genetic programming. In *2nd International Symposium on Logistics and Industrial Informatics, LINDI 2009*, pages 1–6, Linz, Austria, 10-11 September 2009.
- [107] Gabriel Kronberger, Stephan Winkler, Michael Affenzeller, Andreas Beham, and Stefan Wagner. On the success rate of crossover operators for genetic programming with offspring selection. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2009*, volume 5717 of *Lecture Notes in Computer Science*, pages 793–800. Springer Berlin / Heidelberg, 2009.
- [108] William Kruskal. Relative importance by averaging over orderings. *The American Statistician*, 41:6–10, 1987.
- [109] William Kruskal and Ruth Majors. Concepts of relative importance in recent scientific literature. *The American Statistician*, 43:2–6, 1989.
- [110] W. B. Langdon and R. Poli. Fitness causes bloat: Mutation. In Chris Clack, Kanta Vekaria, and Nadav Zin, editors, *ET'97 Theory and Application of Evolutionary Computation*, pages 59–77, University College London, UK, 15 December 1997.
- [111] William B. Langdon. The evolution of size in variable length representations. In *1998 IEEE International Conference on Evolutionary Computation*, pages 633–638, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.
- [112] William B. Langdon. Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines*, 1(1/2):95–119, April 2000.
- [113] William B. Langdon. A SIMD interpreter for genetic programming on GPU graphics cards. Technical Report CSM-470, Department of Computer Science, University of Essex, Colchester, UK, 3 July 2007.
- [114] William B. Langdon and Bernard F. Buxton. Genetic programming for mining dna chip data from cancer patients. *Genetic Programming and Evolvable Machines*, 5(3):251–257, September 2004.

- [115] William B. Langdon and Andrew P. Harrison. GP on SPMD parallel graphics hardware for mega bioinformatics data mining. *Soft Computing*, 12(12):1169–1183, October 2008. Special Issue on Distributed Bioinspired Algorithms. Submitted 26 Sept 2007.
- [116] William B. Langdon and Riccardo Poli. Fitness causes bloat. Technical Report CSRP-97-09, University of Birmingham, School of Computer Science, Birmingham, B15 2TT, UK, 24 February 1997.
- [117] William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.
- [118] Vincent Lemair, Carine Hue, and Olivier Bernier. *Data Mining in Public and Private Sectors: Organizational and Government Applications*, chapter Correlation Analysis in Classifiers, pages 204–218. IGI Global, 2010.
- [119] Vincent Lemaire and Fabrice Clerot. An input variable importance definition based on empirical data probability and its use in variable selection. In *IEEE International Joint Conference on Neural Networks, 2004*, volume 2, pages 1375–1380, July 2004.
- [120] Vincent Lemaire, Raphaël Feraud, and Nicolas Voisine. Contact personalization using a score understanding method. In *International Conference On Neural Information Processing (ICONIP)*, Hong-Kong, October 2006.
- [121] Jin Li and Edward P. K. Tsang. Investment decision making using FGP: A case study. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1253–1259, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.
- [122] Richard H. Lindeman, Peter F. Merenda, and Ruth Z. Gold. *Introduction to Bivariate and Multivariate Analysis*. Scott Foresman, Glenview, IL, 1980.
- [123] Xueyi Liu, Yikang Wang, and Wenhui Wang. Prediction of silicon content in hot metal based on bayesian network. *International Conference on Natural Computation*, 5:446–450, 2007.
- [124] Lennart Ljung. Perspectives on system identification. *Annual Reviews in Control*, 34(1):1–12, 2010.
- [125] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:128–137, 1982. Technical Note, Bell Laboratories.
- [126] Marco Lotz and Sara Silva. Application of genetic programming classification in an industrial process resulting in greenhouse gas emission reductions. In Cecilia Di Chio, Anthony Brabazon, Gianni A. Di Caro, Marc Ebner, Muddassar Farooq, Andreas Fink, Jorn Grahl, Gary Greenfield, Penousal Machado, Michael

- O'Neill, Ernesto Tarantino, and Neil Urquhart, editors, *Applications of Evolutionary Computation*, volume 6025 of *Lecture Notes in Computer Science*, pages 131–140. Springer, 2010.
- [127] Sean Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, September 2000.
 - [128] Sean Luke and Liviu Panait. Lexicographic parsimony pressure. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
 - [129] Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, 2006.
 - [130] Prasanta Chandra Mahalanobis. On the generalised distance in statistics. *National Institute of Sciences of India*, 2(1), April 1936.
 - [131] John Paul Marney, D. Miller, Colin Fyfe, and Heather F. E. Tarbert. Risk adjusted returns to technical trading rules: a genetic programming approach. In *7th International Conference of Society of Computational Economics*, Yale, 28-29 June 2001.
 - [132] James Martens. Deep learning via hessian-free optimization. In *Proc. of the 27th International Conference on Machine Learning, Haifa, Isreal*, 2010.
 - [133] Paul Massey, John A. Clark, and Susan Stepney. Evolution of a human-competitive quantum fourier transform algorithm using genetic programming. In Hans-Georg Beyer, Una-May O'Reilly, Dirk V. Arnold, Wolfgang Banzhaf, Christian Blum, Eric W. Bonabeau, Erick Cantu-Paz, Dipankar Dasgupta, Kalyanmoy Deb, James A. Foster, Edwin D. de Jong, Hod Lipson, Xavier Llorca, Spiros Mancoridis, Martin Pelikan, Guenther R. Raidl, Terence Soule, Andy M. Tyrrell, Jean-Paul Watson, and Eckart Zitzler, editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1657–1663, Washington DC, USA, 25-29 June 2005. ACM Press.
 - [134] Julian F. Miller. Evolution of digital filters using a gate array model. In Riccardo Poli, Hans-Michael Voigt, Stefano Cagnoni, David Corne, George D. Smith, and Terence C. Fogarty, editors, *Proceedings of the First European Workshops on Evolutionary Image Analysis, Signal Processing and Telecommunications*, volume 1596 of *LNCIS*, pages 17–30, Goteborg, Sweden, 26-27 May 1999. Springer-Verlag. ISBN 3-540-65837-8.
 - [135] Julian F. Miller and Peter Thomson. Cartesian genetic programming. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and

- Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 15–16 April 2000. Springer-Verlag. ISBN 3-540-67339-3.
- [136] David J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, 7 May 1993.
 - [137] Gearoid Murphy and Conor Ryan. A simple powerful constraint for genetic programming. In *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 146–157, Naples, 26–28 March 2008. Springer. doi: doi:10.1007/978-3-540-78671-9_13.
 - [138] Gearoid Murphy and Conor Ryan. Exploiting the path of least resistance in evolution. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1251–1258, Atlanta, GA, USA, 12–16 July 2008. ACM.
 - [139] Nadia Nedjah, Enrique Alba, and Luiza de Macedo Mourelle, editors. *Parallel Evolutionary Computations*. Springer, 2006.
 - [140] Christopher J. Neely. Risk-adjusted, ex ante, optimal technical trading rules in equity markets. *International Review of Economics and Finance*, 12(1):69–87, Spring 2003.
 - [141] Christopher J. Neely and Paul A. Weller. Technical trading rules in the european monetary system. *Journal of International Money and Finance*, 18(3):429–458, 1999.
 - [142] Christopher J. Neely and Paul A. Weller. Technical analysis and central bank intervention. *Journal of International Money and Finance*, 20(7):949–970, December 2001.
 - [143] Christopher J. Neely, Paul A. Weller, and Rob Dittmar. Is technical analysis in the foreign exchange market profitable? A genetic programming approach. *The Journal of Financial and Quantitative Analysis*, 32(4):405–426, December 1997.
 - [144] Nikolay Y. Nikolaev and Hitoshi Iba. Genetic programming of polynomial models for financial forecasting. In Shu-Heng Chen, editor, *Genetic Algorithms and Genetic Programming in Computational Finance*, chapter 5, pages 103–123. Kluwer Academic Press, 2002.
 - [145] Peter Nordin. A compiling genetic programming system that directly manipulates the machine code. In Kenneth E. Kinneer, Jr., editor, *Advances in Genetic Programming*, chapter 14, pages 311–331. MIT Press, 1994.

- [146] Peter Norvig. *Paradigms of artificial intelligence programming: case studies in common LISP*. Morgan Kaufmann, 1992.
- [147] Michael O’Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, volume 4 of *Genetic programming*. Kluwer Academic Publishers, 2003. ISBN 1-4020-7444-1. URL <http://www.wkap.nl/prod/b/1-4020-7444-1>.
- [148] Michael O’Neill, Leonardo Vanneschi, Steven Gustafson, and Wolfgang Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3–4):339–396, September 2010.
- [149] Una-May O’Reilly and Franz Oppacher. Hybridized crossover-based search techniques for program discovery. In *Proceedings of the 1995 World Conference on Evolutionary Computation*, volume 2, pages 573–578, Perth, Australia, 29 November - 1 December 1995. IEEE Press. ISBN 0-7803-2759-4.
- [150] Michael Orlov and Moshe Sipper. Finch: A system for evolving java (bytecode). In Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, pages 1–16. Springer New York, 2011. ISBN 978-1-4419-7747-2.
- [151] Mouloud Oussaidène, Bastien Chopard, Olivier V. Pictet, and Marco Tomassini. Parallel genetic programming and its application to trading model induction. *Parallel Computing*, 23(8):1183–1198, August 1997.
- [152] Oxford Dictionary of English. *The Oxford Dictionary of English*. Oxford University Press, 2nd edition, 2005.
- [153] Liviu Panait and Sean Luke. Alternative bloat control methods. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund Burke, Paul Darwen, Dipankar Dasgupta, Dario Floreano, James Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andy Tyrrell, editors, *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 630–641, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.
- [154] Gisele L. Pappa and Alex. A. Freitas. *Automating the Design of Data Mining Algorithms: an Evolutionary Computation Approach*. Springer, 2010. xiii + 187 pages.
- [155] J. G. Peacey and W. G. Davenport. *The Iron Blast Furnace: Theory and Practice*. Pergamon Press, 1979.
- [156] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

- [157] Donald B. Percival and Andrew T. Walden. *Spectral Analysis for Physical Applications*. Cambridge University Press, 1998.
- [158] Frank Pettersson, Henrik Saxen, and Jan Hinnelä. A genetic algorithm evolving charging programs in the ironmaking blast furnace. *Materials and Manufacturing Processes*, 20(3):351–361, 2005.
- [159] Riccardo Poli. Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, B15 2TT, UK, September 1996.
- [160] Riccardo Poli. Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, B15 2TT, UK, September 1996.
- [161] Riccardo Poli. Covariant tarpeian method for bloat control in genetic programming. In Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva, editors, *Genetic Programming in Theory and Practice VIII*, pages 71–89. Springer, 2011.
- [162] Riccardo Poli and Nicholas F. McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2):169–206, June 2003.
- [163] Riccardo Poli and Nicholas F. McPhee. Covariant parsimony pressure for genetic programming. Technical Report CES-480, Department of Computing and Electronic Systems, University of Essex, UK, 2008.
- [164] Riccardo Poli, William B. Langdon, and Stephen Dignum. On the limiting distribution of program sizes in tree-based genetic programming. In Marc Ebner, Michael O’Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 193–204, Valencia, Spain, 11-13 April 2007. Springer.
- [165] Riccardo Poli, William B. Langdon, and Nicholas F. McPhee. *A field guide to genetic programming*. Published via <http://lulu.com> and freely available at <http://www.gp-field-guide.org.uk>, 2008. (With contributions by J. R. Koza).
- [166] Riccardo Poli, Leonardo Vanneschi, William B. Langdon, and Nicholas F. McPhee. Theoretical results in genetic programming: the next ten years? *Genetic Programming and Evolvable Machines*, 11(3–4):285–320, 2010.
- [167] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2002.
- [168] George R. Price. Selection and covariance. *Nature*, 227:520–521, August 1970.

- [169] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.
- [170] V. R. Radhakrishnan and A. R. Mohamed. Neural networks for the identification and control of blast furnace hot metal quality. *Journal of Process Control*, 10(6): 509 – 524, 2000.
- [171] Ingo Rechenberg. *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fommann-Holzboog, Stuttgart, 1973.
- [172] Marko Robnik-Sikonja and Igor Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20: 589–600, 2008.
- [173] Frank Rosenblatt. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan Books, 1962.
- [174] Kumara Sastry. *Genetic Algorithms and Genetic Programming for Multiscale Modeling: Applications in Materials Science and Chemistry and Advances in Scalability*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, September 2007. IlliGAL Report No. 2007019.
- [175] Abraham Savitzky and Marcel J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.
- [176] Henrik Saxen, Frank Pettersson, and Kiran Gunturu. Evolving nonlinear time-series models of the hot metal silicon content in the blast furnace. *Materials and Manufacturing Processes*, 22(5):577–584, 2007. ISSN 1042-6914.
- [177] Henrik Saxen, Frank Pettersson, and Matias Waller. Mining data from a metallurgical process by a novel neural network pruning method. In Bartłomiej Beliczynski, Andrzej Dzieliński, Marcin Iwanowski, and Bernardete Ribeiro, editors, *Adaptive and Natural Computing Algorithms*, volume 4432 of *Lecture Notes in Computer Science*, pages 115–122. Springer Berlin / Heidelberg, 2007.
- [178] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, April 2009.
- [179] Michael Schmidt and Hod Lipson. A rebuttal to Christopher Hillar and Friedrich Sommer’s comment on distilling laws from data. www, July 24 2009.
- [180] Michael Schmidt and Hod Lipson. Symbolic regression of implicit equations. In Rick Riolo, Una-May O’Reilly, and Trent McConaghy, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, pages 73–85. Springer US, 2010.

- [181] Michael D. Schmidt and Hod Lipson. Co-evolving fitness predictors for accelerating and reducing evaluations. In Rick L. Riolo, Terence Soule, and Bill Worzel, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 17, pages –. Springer, Ann Arbor, 11-13 May 2006.
- [182] Peter Schmöle and Hans Bodo Lüngen. Einsatz von vorreduzierten Stoffen im Hochofen: metallurgische, ökologische und wirtschaftliche Aspekte. *Stahl und Eisen*, 4(127):47–56, 2007.
- [183] Marc Schoenauer, Michele Sebag, Francois Jouve, Bertrand Lamy, and Habibou Maitournam. Evolutionary identification of macro-mechanical models. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 23, pages 467–488. MIT Press, Cambridge, MA, USA, 1996. ISBN 0-262-01158-1.
- [184] Hans Schoppa. *Was der Hochöfner von seiner Arbeit wissen muss*. Stahleisen, Düsseldorf, 4 edition, 1992.
- [185] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Birkhäuser Verlag, Basel und Stuttgart, 1977.
- [186] Sara Silva and Jonas Almeida. Dynamic maximum tree depth. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O’Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1776–1787, Chicago, 12-16 July 2003. Springer-Verlag.
- [187] Sara Silva and Ernesto Costa. Dynamic limits for bloat control: Variations on size and depth. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund Burke, Paul Darwen, Dipankar Dasgupta, Dario Floreano, James Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andy Tyrrell, editors, *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 666–677, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.
- [188] Sara Silva and Ernesto Costa. Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, 10(2):141–179, 2009.
- [189] Sara Silva and Stephen Dignum. Extending operator equalisation: Fitness based self adaptive length distribution for bloat free GP. In Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner, editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 159–170, Tuebingen, April 15-17 2009. Springer.

- [190] Sara Silva and Leonardo Vanneschi. Operator equalisation, bloat and overfitting: a study on human oral bioavailability prediction. In Guenther Raidl, Franz Rothlauf, Giovanni Squillero, Rolf Drechsler, Thomas Stuetzle, Mauro Birattari, Clare Bates Congdon, Martin Middendorf, Christian Blum, Carlos Cotta, Peter Bosman, Joern Grahl, Joshua Knowles, David Corne, Hans-Georg Beyer, Ken Stanley, Julian F. Miller, Jano van Hemert, Tom Lenaerts, Marc Ebner, Jaume Bacardit, Michael O'Neill, Massimiliano Di Penta, Benjamin Doerr, Thomas Jansen, Riccardo Poli, and Enrique Alba, editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1115–1122, Montreal, 8-12 July 2009. ACM.
- [191] Sara Silva, Maria Vasconcelos, and Joana Melo. Bloat free genetic programming versus classification trees for identification of burned areas in satellite imagery. In Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Aniko Ekart, Anna I. Esparcia-Alcazar, Chi-Keong Goh, Juan J. Merelo, Ferrante Neri, Mike Preuss, Julian Togelius, and Georgios N. Yannakakis, editors, *EvoIASP*, volume 6024 of *LNCIS*, Istanbul, 7-9 April 2010. Springer.
- [192] Sabine Slaby, Dietmar Andahazy, Franz Winter, Christoph Feilmayr, and Thomas Bürgler. Reducing ability of CO and H₂ of gases formed in the lower part of the blast furnace by gas and oil injection. *ISIJ International*, 46(7):1006–1013, 2006.
- [193] Guido F. Smits and Mark Kotanchek. Pareto-front exploitation in symbolic regression. In Una-May O'Reilly, Tiny Yu, Rick Riolo, and Bill Worzel, editors, *Genetic Programming in Theory and Practice II*, pages 283–299. Springer, 2005.
- [194] Terence Soule and James A. Foster. Code size and depth flows in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 313–320, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.
- [195] Terence Soule and James A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference on Evolutionary Computation*, pages 781–786, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.
- [196] Charles Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.
- [197] Lee Spector. Towards practical autoconstructive evolution: Self-evolution of problem-solving genetic programming systems. In Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, chapter 2, pages 17–34. Springer, 2011.

- [198] Lee Spector and Alan Robinson. Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40, March 2002. ISSN 1389-2576.
- [199] Stahlinstitut VDEh / Wirtschaftsvereinigung Stahl (WV), editor. *Jahrbuch Stahl 2008, Band 1*. Stahleisen, 2007.
- [200] Jean Steinier, Yves Termonia, and Jules Deltour. Comments on smoothing and differentiation of data by simplified least square procedure. *Analytical Chemistry*, 11(44):1906–1909, 1972.
- [201] Thomas Sterling. Beowulf-class clustered computing: Harnessing the power of parallelism in a pile of PCs. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, page 883, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann. Invited talk.
- [202] Julius H. Strassburger, Dwight C. Brown, Terence E. Dancy, and Robert L. Stephenson, editors. *Blast furnace - theory and practice*. Gordon and Breach Science Publishers, New York, 1969. Second Printing August 1984.
- [203] Carolin Strobl and Achim Zeileis. Danger: High power! - exploring the statistical properties of a test for random forest variable importance. In P. Brito, editor, *COMPSTAT 2008 - Proceedings in Computational Statistics*, volume 2, pages 59–66, Heidelberg, Germany, 2008. Physica Verlag.
- [204] Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(25), January 2007.
- [205] Carolin Strobl, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis. Conditional variable importance for random forests. Technical Report 23, Department of Statistics, University of Munich, 2008.
- [206] Carolin Strobl, Torsten Hothorn, and Achim Zeileis. Party on! a new, conditional variable importance measure for random forests available in the party package. Technical Report 50, Department of Statistics, University of Munich, 2009.
- [207] Erik Strumbelj and Igor Kononenko. Towards a model independent method for explaining classification for individual instances. In I.-Y. Song, J. Eder, and T.M. Nguyen, editors, *DaWaK 2008*, number 5182 in LNCS, pages 273–282. Springer-Verlag Berlin Heidelberg, 2008.
- [208] Erik Strumbelj and Igor Kononenko. An efficient explanation of individual classifications using game theory. *Journal of Machine Learning Research*, 11:1–18, January 2010.

- [209] Erik Strumbelj, Igor Kononenko, and Marko Robnik-Sikonja. Explaining instance classifications with interactions of subsets of feature values. *Data & Knowledge Engineering*, 68:886–904, 2009.
- [210] Richard S. Sutton, , and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
- [211] G. J. Székely, M. L. Rizzo, and N. K. Bakirov. Measuring and testing independence by correlation of distances. *Annals of Statistics*, 35(6):2769–2794, 2007.
- [212] Astro Teller and David Andre. Automatically choosing the number of fitness cases: The rational allocation of trials. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 321–328, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.
- [213] Henri Theil. *Applied Economic Forecasting*. North-Holland Publishing Company, Amsterdam, 1966.
- [214] Henry Theil. How many bits of information does an independent variable yield in multiple regression? *Statistics and Probability Letters*, 6:107–108, 1987.
- [215] Henry Theil and C. F. Chung. Information-theoretic measures of fit for univariate and multivariate linear regression. *The American Statistician*, 42:249–252, 1988.
- [216] Edward Tsang, Paul Yung, and Jin Li. EDDIE-automation, a decision support tool for financial forecasting. *Decision Support Systems*, 37(4):559–565, 2004. Data mining for financial decision making.
- [217] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, second edition, 2001.
- [218] Edward R. Tufte. *Envisioning Information*. Graphics Press LLC, Cheshire, Connecticut, 2006. Eleventh Printing.
- [219] Shigeru Ueda, Shungo Natsui, Hiroshi Nogami, Jun ichiro Yagi, and Tatsuro Ariyama. Recent progress and future perspective on mathematical modeling of blast furnace. *ISIJ International*, 50(7):914–923, 2010.
- [220] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In *Proc. GECCO’10*, pages 877–884, July 7–11 2010.
- [221] Vladimir Vapnik, Steven E. Golowich, and Alex Smola. Support vector method for function approximation, regression estimation, and signal processing. In *Advances in Neural Information Processing Systems 9*, pages 281–287. MIT Press, 1996.
- [222] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

- [223] Ekaterina Vladislavleva, Guido Smits, and Dick den Hertog. On the importance of data balancing for symbolic regression. *IEEE Transactions on Evolutionary Computation*, 14(7), April 2010.
- [224] Stefan Wagner. *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. PhD thesis, Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria, 2009.
- [225] Wenhui Wang. Application of BP network and principal component analysis to forecasting the silicon content in blast furnace hot metal. *Workshop on Intelligent Information Technology Applications, 2007*, 3:42–46, 2008.
- [226] Yikang Wang and Xiangguan Liu. Prediction of silicon content in hot metal based on the combined model of EMD and SVM. *Workshop on Intelligent Information Technology Applications, 2007*, 1:561–565, 2008.
- [227] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. Automatically finding patches using genetic programming. In Stephen Fickas, editor, *International Conference on Software Engineering (ICSE) 2009*, pages 364–374, Vancouver, May 16-24 2009. URL <http://www.cs.unm.edu/~tnguyen/papers/genprog.pdf>.
- [228] P. A. Whigham. Search bias, language bias, and genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 230–237, Stanford University, CA, USA, 28–31 July 1996. MIT Press.
- [229] Peter Whittle. On the fitting of multivariate autoregressions and the approximate canonical factorization of a spectral density matrix. *Biometrika*, 40:129–134, 1963.
- [230] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.
- [231] Stephan Winkler, Michael Affenzeller, and Stefan Wagner. Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis. *Genetic Programming and Evolvable Machines*, 10(2):111–140, 2009.
- [232] Stephan M. Winkler. *Evolutionary System Identification - Modern Concepts and Practical Applications*. Number 59 in Reihe C - Technik und Naturwissenschaften. Trauner Verlag, Linz, 2008.
- [233] Ian H. Witten and Eibe Frank. *Data mining: practical machine learning tools and techniques*. Morgan Kaufman - Elsevier, 2nd edition, 2005.
- [234] Man Leung Wong and Kwong Sak Leung. *Data Mining Using Grammar Based Genetic Programming and Applications*, volume 3 of *Genetic Programming*. Kluwer Academic Publishers, January 2000. ISBN 0-7923-7746-X.

- [235] Man-Leung Wong, Tien-Tsin Wong, and Ka-Ling Fok. Parallel evolutionary algorithms on graphics processing unit. In David Corne, Zbigniew Michalewicz, Bob McKay, Gusz Eiben, David Fogel, Carlos Fonseca, Garrison Greenwood, Gunther Raidl, Kay Chen Tan, and Ali Zalzala, editors, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2286–2293, Edinburgh, Scotland, UK, 2-5 September 2005. IEEE Press.
- [236] Qizhi Yu, Chongcheng Chen, and Zhigeng Pan. Parallel genetic algorithms on programmable graphics hardware. In Lipo Wang, Ke Chen, and Yew-Soon Ong, editors, *Advances in Natural Computation, First International Conference, ICNC 2005, Proceedings, Part III*, volume 3612 of *Lecture Notes in Computer Science*, pages 1051–1059, Changsha, China, August 27-29 2005. Springer.
- [237] Tina Yu and Shu-Heng Chen. Using genetic programming with lambda abstraction to find technical trading rules. In *Computing in Economics and Finance*, University of Amsterdam, 8-10 July 2004.
- [238] Alexandru-Ciprian Zavoianu. Towards solution parsimony in an enhanced genetic programming process. Master’s thesis, International School Informatics: Engineering & Management, ISI-Hagenberg, Johannes Kepler University, Linz, 2010.
- [239] Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

A.4. Colophon

This thesis has been type set with \LaTeX and \BibTeX . The author used primarily `emacs` to edit the source files. `Subversion` has been used as revision control system for all related files.

The variable relationship graphs in Chapters 6 and 8 have been generated with the Graphviz graph visualization software. The line-charts, scatter-plots, and 3-d surface plots have been generated with the `pgfplots` package by Christian Feuersänger. The result tables have been generated with the `pgfplotstable` package also by Christian Feuersänger. Box-plots in Chapters 6 and 8 and the kernel density estimation plot have been created using R, the free software environment for statistical computing and graphics. Screenshots in Chapters 4 6 show HeuristicLab <http://dev.heuristiclab.com>, a software environment for heuristic optimization developed by members of the research group HEAL. Bitmap graphics have been edited with Paint.NET and converted to postscript format with Inkscape.

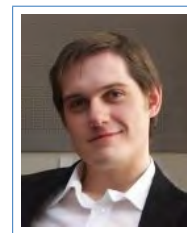
Gabriel Kronberger

Curriculum Vitae

Marienstrasse 4
4020 Linz

☎ mobile +43 650/7614904

✉ gkronber@heuristiclab.com



Personal Data

Position	Research Assistant, Upper Austria University of Applied Sciences, Research Center Hagenberg
Dates of Birth	June 29th, 1981 in Kirchdorf, Austria
Nationality	Austrian
Marital status	Unmarried

Education

1987–1991	Primary School , <i>Volksschule</i> , Micheldorf, Austria.
1991–1995	Secondary School , <i>Bundesrealgymnasium</i> , Kirchdorf, Austria.
1995–1999	High School , <i>Bundesrealgymnasium</i> , Kirchdorf, Austria.
2000–2005	Masters degree in computer science , <i>Johannes Kepler University</i> , Linz, Austria.
2005–2010	PhD studies in engineering sciences , <i>Johannes Kepler University</i> , Linz, Austria.

Experience

Vocational

2003	Web-developer , <i>Ars Electronica Center, Digital Economy</i> , Linz, Austria.
since 2005	Research Assistant , <i>Upper Austria University of Applied Sciences, Research Center Hagenberg</i> , Austria.
since 2006	External Lecturer , <i>Upper Austria University of Applied Sciences, Hagenberg</i> , Austria, Courses in Software Engineering: Introduction to Programming, Elementary Algorithms and Data-structures, Advanced Algorithms and Data-structures, and Distributed and Parallel Software Systems. Courses in Bioinformatics: Numerical Methods in Bioinformatics.

Public Service

1999–2000	Military service , <i>Kremstalker Kaserne</i> , Kirchdorf, Austria.
-----------	--

Publications

Book Chapters

- [1] M. Affenzeller, S. M. Winkler, S. Wagner, A. Beham, G. K. Kronberger, and M. Kofler. Meta-heuristic optimization. In *Hagenberg Research*, pages 103–156. Springer Verlag, 2009.

Conference Articles

- [2] M. Affenzeller, G. K. Kronberger, S. M. Winkler, M. Ionescu, and S. Wagner. Heuristic optimization methods for the tuning of input parameters of simulation models. In *Proc. of International Mediterranean Modelling Multiconference I3M2007*, pages 278–283, Genoa, Italy, October 2007.
- [3] M. Affenzeller, L. Pöllabauer, G. K. Kronberger, E. Pitzer, S. Wagner, S. M. Winkler, A. Beham, and M. Kofler. On-line parameter optimization strategies for metaheuristics. In *Proc. of 22nd European Modeling and Simulation Symposium EMSS 2010*, pages 31–36, Fes, Morocco, October 2010.
- [4] A. Beham, M. Affenzeller, S. Wagner, and G. K. Kronberger. Simulation optimization with HeuristicLab. In *Proc. of the 20th European Modeling and Simulation Symposium*, pages 75–80, Campora San Giovanni, Spain, September 2009.
- [5] M. Kofler, S. Wagner, A. Beham, G. K. Kronberger, and M. Affenzeller. Priority rule generation with a genetic algorithm to minimize sequence dependent setup costs. In Roberto Moreno-Diaz, Franz Pichler, and Alexis Quesada-Arencia, editors, *Computer Aided Systems Theory - EUROCAST 2009, 12th International Conference*, volume 5717 of *Lecture Notes in Computer Science*, pages 817–824. Springer Verlag, October 2009.
- [6] M. Kommenda, G. K. Kronberger, M. Affenzeller, S. M. Winkler, C. Feilmayr, and S. Wagner. Symbolic regression with sampling. In *Proc. of 22nd European Modeling and Simulation Symposium EMSS 2010*, pages 13–18, Fes, Morocco, October 2010.
- [7] M. Kommenda, G. K. Kronberger, S. M. Winkler, M. Affenzeller, S. Wagner, L. Schickmair, and B. Lindner. Application of genetic programming on temper mill datasets. In *Proc. of the IEEE 2nd International Symposium on Logistics and Industrial Informatics (Lindi 2009)*, pages 58–62, Linz, Austria, September 2009.
- [8] G. K. Kronberger and R. Braune. Bandit-based monte-carlo planning for the single-machine total weighted tardiness scheduling problem. In Roberto Moreno Diaz, Franz Pichler, and Alexis Quesada Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2007*, volume 4739 of *Lecture Notes in Computer Science*, pages 837–844. Springer Verlag, December 2007.
- [9] G. K. Kronberger, C. Feilmayr, M. Kommenda, S. M. Winkler, M. Affenzeller, and T. Buerghler. System identification of blast furnace processes with genetic programming. In *Proc. of the IEEE 2nd International Symposium on Logistics and Industrial Informatics (Lindi 2009)*, pages 63–68, Linz, Austria, September 2009.
- [10] G. K. Kronberger and A. Weidenhiller. Automated generation of simulation models from ERP data - an example. In *Proc. FH Science Day 2006*, pages 160–165, Hagenberg, Austria, October 2006.
- [11] G. K. Kronberger, A. Weidenhiller, B. Kerschbaumer, and H. Jodlbauer. Automated simulation model generation for scheduler-benchmarking in manufacturing. In *Proc. of the International Mediterranean Modelling Multiconference (I3M 2006)*, pages 45–50, Barcelona, Spain, October 2006.
- [12] G. K. Kronberger, S. M. Winkler, M. Affenzeller, A. Beham, and S. Wagner. On the success rate of crossover operators for genetic programming with offspring selection. In Roberto Moreno-Diaz, Franz Pichler, and Alexis Quesada-Arencia, editors, *Computer Aided Systems Theory - EUROCAST 2009, 12th International Conference*, volume 5717 of *Lecture Notes in Computer Science*, pages 793–800. Springer Verlag, November 2009.
- [13] G. K. Kronberger, S. M. Winkler, M. Affenzeller, M. Kommenda, and S. Wagner. Effects of mutation before and after offspring selection in genetic programming for symbolic regression. In *Proc. of 22nd European Modeling and Simulation Symposium EMSS 2010*, pages 37–42, Fes, Morocco, October 2010.

- [14] G. K. Kronberger, S. M. Winkler, M. Affenzeller, and S. Wagner. Data mining via distributed genetic programming agents. In *Proc. of the 20th European Modeling and Simulation Symposium*, pages 95–99, Campora San Giovanni, Spain, September 2009.
- [15] G. K. Kronberger, S. M. Winkler, M. Affenzeller, and S. Wagner. On crossover success rate in genetic programming with offspring selection. In Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner, editors, *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 232–243. Springer Verlag, April 2009.
- [16] S. Wagner, M. Affenzeller, A. Beham, G. K. Kronberger, and S. M. Winkler. Mutation effects in genetic algorithms with offspring selection applied to combinatorial optimization problems. In *Proc. of 22nd European Modeling and Simulation Symposium EMSS 2010*, pages 43–48, Fes, Morocco, October 2010.
- [17] S. Wagner, A. Beham, G. K. Kronberger, M. Kommenda, E. Pitzer, M. Kofler, S. Vonolfen, S. M. Winkler, V. Dorfer, and M. Affenzeller. HeuristicLab 3.3: A unified approach to metaheuristic optimization. In *Proceedings of the 7th Spanish Congress on Metaheuristics, Evolutionary and Bioinspired Algorithms (MAEB'10)*, Valencia, Spain, September 2010.
- [18] S. Wagner, G. K. Kronberger, A. Beham, S. M. Winkler, and M. Affenzeller. Model driven rapid prototyping of heuristic optimization algorithms. In Roberto Moreno-Diaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2009, 12th International Conference*, volume 5717 of *Lecture Notes in Computer Science*, pages 729–736. Springer Verlag, December 2009.
- [19] S. Wagner, G. K. Kronberger, A. Beham, S. M. Winkler, and M. Affenzeller. Modeling of heuristic optimization algorithms. In *Proc. of the 20th European Modeling and Simulation Symposium*, pages 106–111, Campora San Giovanni, Spain, September 2009.
- [20] S. Wagner, S. M. Winkler, E. Pitzer, G. K. Kronberger, A. Beham, R. Braune, and M. Affenzeller. Benefits of plugin-based heuristic optimization software systems. In Roberto Moreno Diaz, Franz Pichler, and Alexis Quesada Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2007*, volume 4739 of *Lecture Notes in Computer Science*, pages 747–754. Springer Verlag, November 2007.
- [21] A. Weidenhiller and G. K. Kronberger. New developments in scheduling - improved model and solver approaches. In *Proc. FH Science Day 2006*, pages 153–159, Hagenberg, Austria, October 2006.
- [22] A. Weidenhiller and G. K. Kronberger. Simulation-based comparison of integrated scheduling with MRP and KANBAN. In *Proc. FH Science Day 2007*, pages 139–143, Wels, Austria, October 2007.
- [23] S. M. Winkler, M. Affenzeller, G. K. Kronberger, M. Kommenda, and S. Wagner. Using genetic programming in nonlinear model identification. In *Proc. of Workshop on Identification in Automotive 2010*, Linz, Austria, July 2010.
- [24] S. M. Winkler, M. Affenzeller, G. K. Kronberger, S. Wagner, W. Jacak, and H. Stekel. Feature selection in the analysis of tumor marker data using evolutionary algorithms. In *Proc. of 22nd European Modeling and Simulation Symposium EMSS 2010*, pages 1–6, Fes, Morocco, October 2010.
- [25] S. M. Winkler, M. Affenzeller, S. Wagner, and G. K. Kronberger. Analysis of patient data and evolutionary design of virtual sensors for diseases. In *Proc. of the 3rd Symposium of Austrian Universities of Applied Sciences*, pages 56–61, FH Kärnten, Villach, Austria, April 2009.
- [26] S. M. Winkler, M. Affenzeller, S. Wagner, G. K. Kronberger, and A. Beham. Evolutionäres design von virtuellen sensoren. In *Proc. of the Industrial Symposium Mechatronics 2007 on Sensorics*, pages 166–175, Linz, Austria, October 2007.
- [27] S. M. Winkler, M. Affenzeller, S. Wagner, G. K. Kronberger, and M. Kommenda. Analysis of structural similarities of mathematical models in the context of genetic programming. In *Proc. of the 4th Symposium of Austrian Universities of Applied Sciences*, pages 315–320, Pinkafeld, Austria, April 2010.