A Modular Genetic Programming System

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r I n g e n i e u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Oliver Flasch

Dortmund

2015

Tag der mündlichen Prüfung
6. Mai 2015

Dekan
Prof. Dr.-Ing. Gernot A. Fink

Gutachter
Prof. Dr. Günter Rudolph
Prof. Dr. Thomas Bartz-Beielstein

Acknowledgments

During my work on this thesis, I had the pleasure to benefit from the advice and support from many friends and colleagues.

Prof. Dr. Thomas Bartz-Beielstein, my thesis advisor, granted me the freedom to pursue my own ideas, while always being available with his encouragement, support, and detailed knowledge. The great working atmosphere at his rapidly growing SPOTSeven group at Cologne University of Applied Sciences has been an important factor in the successful completion of this thesis. I am looking forward to continuing my work there. Prof. Dr. Günter Rudolph, my supervisor at TU Dortmund, generously shared his detailed knowledge on evolutionary computation and provided valuable guidance during many enjoyable meetings.

Olaf Mersmann supported me with his comprehensive knowledge of computational statistics and provided the emoa software package for evolutionary multi-objective optimization, a key component of RGP's selection operators. Additionally, he contributed excellent and highly efficient code to RGP's candidate solution evaluator. Martin Zaefferer supported this work with his in-depth knowledge on Kriging and with many fruitful discussions. Dr. Wolfgang Kantschik introduced me to the practical aspects of Genetic Programming during my time at his company, Dortmund Intelligence Project GmbH. Dr. Boris Naujoks helped with his expertise in multi-objective optimization and with his extensive network of contacts in the scientific community. During her time at SPOTSeven, Dr. Katya Vladislavleva contributed important ideas for the design of modern GP systems. Jörg Stork and Tobias Brandt made valuable contributions to RGP during their time as students at SPOTSeven. RGP greatly profited from feedback of its users, many of which contributed new perspectives, test cases, or application examples.

I am obliged to Thomas Will of Steinmüller Engineering GmbH and to Christian Jung for providing ideas, data, and extensive support for the AppDust and AppSteel case studies respectively. I am grateful to Prof. Dr. Gabriele Kern-Isberner and Prof. Dr. Peter Buchholz for agreeing to survey this thesis on very short notice.

This work would not have been possible without financial support from the Bundesministerium für Bildung und Forschung (BMBF) in form of the grants FIWA (FKZ 17N2309) and MCIOP (FKZ 17N0311).

Most importantly, I am grateful to my family, Katja and Sarah, for their support and for reminding me that there is more to life than genetic programming.

Contents

List of Figures

List of Tables

1 Introduction

Genetic Programming (GP) is an evolutionary algorithm for the automatic discovery of symbolic expressions that encode solutions to a user-defined task. An initial population of symbolic expressions is successively refined by mutation and recombination until a solution of sufficient quality is found. In its general form, the symbolic expressions in a GP population encode executable computer programs, represented as parse trees of a Turing complete programming language. Readability and understandability of these programs is not an explicit concern and also often poor, especially if high quality solutions to difficult problems are sought.

This work has a slightly different focus by concentrating on data-driven discovery of clear, readable, and compact solutions. This is achieved through a multi-objective GP search heuristic implemented by means of a modular GP system integrated into the R environment for statistical computing. Real-world applications discussed in this work include the induction of efficient and accurate models for roll train performance in steel production and the induction of fast meta-models for geometry optimization of cyclone dust separators. In these application domains, expressive and specific formalisms are common to communicate solutions in a clear and concise way.

1.1 Motivation

Main motivation of this work is the advancement of GP in real-world application areas. Therefore, this work is of empirical rather than theoretical nature. The field of GP applications is very broad, making it impossible to describe every possible application in sufficient detail. Because of its practical focus, this work confines itself on a subset of application areas that are already known to be practical for GP at the time of writing, first of all symbolic regression. Nonetheless, the formal framework introduced in this work is general enough to apply to all GP application areas, some of which are subject to detailed exposition in the form of tutorials.

Most text book introductions to the field of GP are not tied to a specific implementation, making them universally useful, but imposing the burden of translating general concepts to the specifics of

an implementation to the potential GP user. Examples include the text books by Koza [1992][1] and Poli et al. [2008].[2] This work takes a different approach. All GP methods and techniques discussed are accompanied by thoroughly documented implementations that can be directly employed by the reader. Together, these implementations constitute the RGP system, an open and modular GP system that is capable enough for real-world application.

Another important motivation for this work is to provide a first independent and systematic study of the performance impact of modern, i.e., multi-objective, GP search heuristics. Goal of this study is to identify what components of a modern GP system are relevant to attain good results, and how these components should be parameterized.

[1] Koza's work provides source code in a now obsolete dialect of the LISP programming language.

[2] Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, MA; and Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza)

1.1.1 *Audience*

The intended audience of this work mainly consists of two groups. The first group are researches in the fields of GP, computational intelligence (CI), machine learning, and neighboring fields of statistics interested in technical details and performance characteristics of modern GP algorithms. The second group consists of practitioners from academia and industry looking for an in-depth introduction to GP and a guide to its application in real-world tasks.

1.2 *Contributions*

This thesis offers three main contributions to the state-of-the art in GP systems:

- *A Modular GP System* The conceptual and technical foundation of the thesis is provided by RGP[3], a modular and self-contained GP system based on and integrated with the R environment for statistical computing. An important feature of RGP's design is the clear separation of search space from search heuristic, enabling detailed studies of the influence of GP algorithm components on GP algorithm performance. RGP provides implementations of classical as well as of state-of-the-art GP algorithm components, including multi-objective selection operators to prevent bloat and provide concise solution formulae, dynamic age-layering and population diversity preservation methods to prevent stagnation in long GP runs and allow the solution of complex real-world tasks, an expressive type system for the concise definition of domain-specific formalisms, and modern tools for post-processing and visualizing GP run results. In its default configuration, RGP is shown to equal or surpass the performance of existing state-of-the-art GP systems, such as ECJ. Nonetheless, its main value stems from its modular design, allowing to combine algorithm components freely without programming or complex reconfiguration.

[3] The name "RGP" may be understood as an abbreviation for "R-based genetic programming".

- *A Comprehensive Empirical Analysis of Modern GP Heuristics* Based on the modular GP system RGP and the methodology of sequential parameter optimization (SPO), a framework for reproducible empirical research in GP has been developed. Statistical tools to understand GP algorithms and heuristics and their interaction with problems of varying difficulty are provided. This framework is then employed in a comprehensive and reproducible empirical analysis of the effects of GP algorithm components on GP performance. In addition to classical GP algorithms, the effect of modern methods, such as multi-objective selection operators and diversity preservation techniques, are also examined. This study focuses on the important GP application of symbolic regression. In this context, result generalizability is improved by using randomized test problems of scalable difficulty. The results of this analysis reveal the beneficial effect of modern GP search heuristics on GP performance compared to traditional GP algorithms. The effects and interactions of several GP algorithm parameters are also analyzed and recommendations for good parameter settings are provided.
- *New Industrial Applications for GP* Finally, the effectiveness of the RGP system is validated in two case studies based on real-world industrial applications. Both applications belong to the class of regression problems from mechanical and process engineering. The first application involves the induction of predictive control models for roll trains in steel production. The second application involves the induction of meta-models for optimizing cyclone dust separator geometries. In both applications, models must be run-time efficient and accurate. Easily understandable, simple models that are easy to deploy offer an additional benefit. In both applications, the time needed for initial model induction can be neglected. A comparison with traditional and modern regression methods shows that RGP-models offer comparable or superior performance in both applications, while adding the additional benefit of understandable and easy to deploy models.

On a methodological level, this work makes the following contributions:

- *Reproducible Research* In the context of this work, the term reproducibility means the practical ability to reproduce the results of an experiment or of an entire study. To achieve reproducibility, all raw data, as well as all software tools, should be available to the general public. The Science Code Manifesto[4] defines additional principles to adhere to for making scientific software accessible and enabling result reproduction through a third party. The authors of this work adhere to these principles by providing documented source code for all experiments for download and keeping it available in the future. Additionally, the raw data for the real-world case studies described in this thesis are available for download under an open source licence.

[4] Barnes, N. et al. (2013). Science code manifesto. http://sciencecodemanifesto.org/ (retrieved 20.02.2015)

- *Open Source Software* In addition to experiment source code and raw data, reproducibility is simplified further by the exclusive use of open source software. RGP, the statistical platform R, as well as all tools used in experiment setup, result analysis and visualization are available for download under an open source licence. Web links to relevant experiment source code and tools are provided when appropriate, throughout the rest of the thesis.

The primary motivation of this work was to provide a practical system applicable to real-world problems. This consideration informed all design decisions, not only on a superficial level, but also on the conceptual and implementation levels. A practical GP system should be easy to understand, easy to configure, and easy to extend if necessary. Most importantly, it should create solutions that are easily understandable and verifiable.

A successful first attempt is provided by this work, in form of the RGP system. RGP is not only a useful system for academic research, but also already proved its effectiveness in several practical applications. Its implementation is sufficiently efficient for the solution of large real-world problems. It is thoroughly documented and approachable through a modern graphical user interface. These traits make RGP useful for experts with diverse scientific and industrial backgrounds.[5]

[5] Comprehensive reference documentation and tutorials for RGP are available online at `http://cran.r-project.org/web/packages/rgp`. Section 3.3 contains additional tutorials, both on basic and on advanced topics.

1.3 Thesis Outline

The remainder of this work is structured as follows. Chapter 2 provides the formal framework for EAs and GP that the rest of this thesis is build upon. Chapter 3 introduces RGP, the modular GP system that provides the basis for most experimental work described in this thesis. Chapter 4 introduces statistical methods for conducting principled empirical research in the field of GP. Based on these foundations, Chapter 5 then proceeds with a detailed study of the performance impact of different components of modern GP search heuristics and their parameterization. Chapter 6 compares the performance of RGP with other state-of-the-art GP systems and provides two detailed case studies of real-world applications. A summary and an outlook to further research are given in Chapter 7.

Figure 1.1 shows different paths a reader could choose through the material presented in this work, taking dependencies between chapters into account.

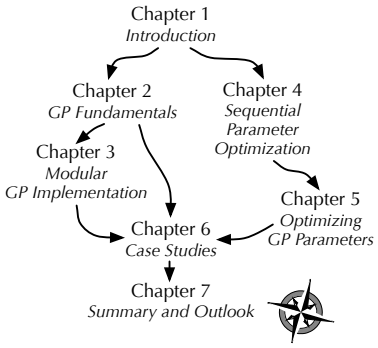


Figure 1.1: Thesis Roadmap. This map shows paths a reader could choose through the chapters of this work, taking dependencies of the presented material into account.

2
Genetic Programming Fundamentals

This chapter introduces the formal framework of GP that the RGP system, as well the empirical work discussed in later chapters, is built upon. After a informal overview of GP, its most important application areas, and its history, the concept of an abstract EA is introduced. On this basis, GP genotypes, phenotypes, fitness, and search operators are defined. The concepts genotypic and phenotypic GP search spaces are introduced next, followed by techniques for defining valid regions in these search spaces. Finally, a selection of classical and modern GP search heuristics is described.

2.1 A Bird's-Eye View of GP

GP is a collection of techniques from evolutionary computing (EC) for the automatic generation of computer programs that perform a user-defined task. Starting with a high-level problem definition in the form of a function associating a candidate solution with a numerical fitness value, GP creates a population of random computer programs that are progressively refined through variation and selection until a satisfactory solution is found. As such, GP is the application of an evolutionary search heuristic[1] to the problem of program synthesis, qualifying as a method for inductive programming.[2] This section provides a short introduction to the most important aspects of GP. For an in-depth introduction, see Poli et al. [2008].[3]

Figure 2.1 shows the control flow of a typical GP system. After creating a population of initial solutions, the algorithm enters the main loop of an evolutionary search routine. The fitness of each solution is then evaluated. Based on their numerical fitness values, solutions are selected for variation and recombination, or removed from the population. The main loop is repeated until a termination criterion is met, after which results are analyzed and presented to the user. In practice, there are many possible implementations for each step in this general scheme, and variations in the order of the steps. For example, selection can take place before and/or after variation and recombination.

[1] cf. the concept of "the survival of the fittest" from Darwin's theory of evolution

[2] Kitzelmann, E. (2010). Inductive programming: A survey of program synthesis techniques. In Schmid, U. et al., editors, *Approaches and Applications of Inductive Programming*, volume 5812 of *Lecture Notes in Computer Science*, pages 50–73. Springer, Berlin

[3] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza)

The idea that GP is the application of EC search heuristics to pro-

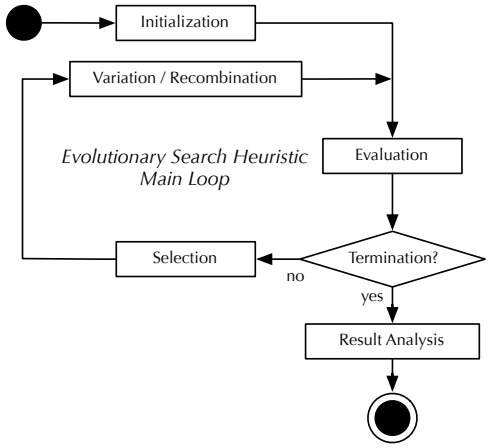


Figure 2.1: Control flow in a typical GP system. After creating an initial population, the algorithm enters its main loop. The fitness of each solution is evaluated. Based on fitness value, solutions are selected for variation and recombination, or removed from the population. The main loop is repeated until a termination criterion holds, after which results can be analyzed.

gram synthesis hints at the distinction of search strategy and search space, whose importance will become apparent in the course of this work. Keeping the concepts of problem definition, search space definition, and search heuristics separate, both in theory and in implementation, opens up exciting possibilities. By changing the search space definition, solutions represented in different formalisms will be generated based on the same problem definition via the same search heuristic. By tuning or changing the search heuristic, satisfactory solutions from the same formalism may be generated more efficiently for a certain problem domain. It is now easily possible to step outside the framework of GP by using search heuristics from different fields of optimization research, e.g., combinatorial optimization. Depending on search space size, the search heuristic could even be a brute-force exhaustive search implemented as a data-parallel program.

In Figure 2.1, the evaluation step is defined by the problem definition, the initialization and variation/recombination steps are defined by the search space, and the selection step, as well as the specific order of the steps, are defined by the search heuristic. The result analysis step is influenced by the search space as well as by the problem definition and is often performed interactively by using techniques from the field of exploratory data analysis (EDA) and statistics.

The term "computer program" in the definition of GP warrants further inspection. Conceptually, a computer program is a term in a formal language, i.e., a symbolic expression.[4] A formal language is a well-defined set of strings of symbols. In the framework of GP, as in the framework of formal languages, no assumptions on the semantics of the symbolic expressions used is made. The only requirements are that 1) there is an algorithm to create, mutate and recombine terms of the formal language, and 2) that there is an algorithm to associate a numerical fitness value with terms of the formal language. This freedom allows GP to be used in areas exceeding what is normally thought of program synthesis, including, for example, the synthesis of electronic circuits or automatic

[4] Pierce, B. C. (2002). *Types and Programming Languages*. MIT Press, Cambridge, MA

theorem proving. [5]

On the other hand, this freedom can also pose problems for the practical applicability of GP. Because the field of possible applications is so broad, implementing a practical, meaning both effective and efficient, yet completely general GP system is a major undertaking. This is mainly because of the theoretical fact and practical observation that search heuristics have to be tuned to certain problem classes to be efficient enough for practical application.[6] The work-around for this problem chosen in this thesis is to first provide a modular framework for GP including a modular software implementation. Then, modules for search heuristics and problem definitions[7] are provided for problem classes and application areas deemed to be both well-suited for state-of-the art GP and of high practical relevance. The framework is made available in an easily accessible form, enabling users to contribute modules to enhance the system's universality.

Figure 2.2 illustrates these concepts as a simple diagram. Given a problem definition in the form of a fitness function, a search heuristic (which is in practice often fixed and specific to the GP system implementation) and a formal language (a formalism), GP will evolve symbolic expressions (formulas) of increasingly better fitness.

[5] Koza, J. R. and Bennett III, F. H. (1999). Automatic synthesis, placement, and routing of electrical circuits by means of genetic programming. In Spector, L. et al., editors, *Advances in Genetic Programming 3*, pages 105–134. MIT Press, Cambridge, MA; and Nordin, P. and Banzhaf, W. (1997). Genetic reasoning: Evolving proofs with genetic search. In Koza, J. R. et al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 255–260, San Francisco, CA. Morgan Kaufmann

[6] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82

[7] implemented as fitness function generators

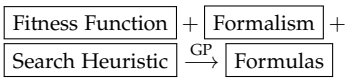


Figure 2.2: General GP diagram. Given a fitness function (the problem definition), a formalism (a formal language), and a search heuristic, GP evolves formulas (symbolic expressions) with increasingly better fitness values.

In many practical applications, GP is used to derive symbolic expressions representing models based on data. These applications include the data-driven induction of models for regression and classification. In the framework of machine learning and data mining, a GP system configured to solve data-driven modeling tasks can be regarded as a method for supervised learning, enabling the transfer of techniques for the analysis of machine learning algorithms to the field of GP.[8]

An important advantage of data-driven GP is that no prior knowledge concerning the model structure is needed. The representation of models as terms of a formal language (symbolic expressions) makes them accessible to human reasoning and symbolic computation, which is another advantage of GP. In this perspective, GP can be seen as a method for discovering symbolic representations of scientific laws[9] hidden in numerical sub-symbolic form as fixed relations in data sets.

This concept is concisely visualized in Figure 2.3: Given a data set, a search heuristic, and a formal language (a formalism), data-driven GP induces symbolic expressions representing fixed relations inherent in the data set (formulas).

[8] Hastie, T. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer, New York, 2nd edition

[9] This includes natural laws as well as functional dependencies of systems studied in economics or sociology.

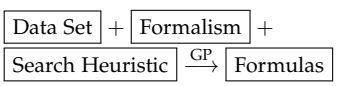


Figure 2.3: Data-driven GP diagram. Given a data set, a search heuristic, and a formalism (a formal language), GP induces formulas (symbolic expressions) representing fixed relations in the data set.

The main drawback of GP is its high computational cost, due to the potentially infinitely large search space of symbolic expressions. On the other hand, the good scalability of evolutionary search heuristics through parallelization, combined with the sustained performance increase of high performance parallel computers,

recently enabled the practical application of GP in many real-world application domains.

2.2 *Application Areas*

As already mentioned in Section 2.1, the set of possible application areas of GP is very large, rendering a complete survey impossible in the confines of this work. As a compromise, this section focuses on areas where GP is successfully employed today and that are also deemed of high practical importance. The majority of these areas fit into the to category of data-driven GP. Unsurprisingly, problems from these areas also belong to the main application area of RGP in its current form and will be discussed in depth by means of case studies in Chapter 6.

2.2.1 *Symbolic Regression*

Symbolic regression is probably the best known application for data-driven GP. Given a set of measurement data divided into dependent and independent variables[10], symbolic regression can discover the functional relationship between dependent and independent variables. This relationship is represented as a symbolic expression, which can be used to gain insight into the data-generating process or system (system identification), and as a model to predict the values of the dependent variable for unseen values of independent variables (inter- and extrapolation).

Figure 2.4 provides a simple example, showing how symbolic regression via GP is used to find the governing physical law of a damped oscillator (e.g. a pendulum). Input data is generated by sampling the position of a swinging pendulum at fixed time intervals, until the pendulum comes to rest. [11] The only independent variable in this example is time, the dependent variable being the pendulum position, represented as an angle. As a fitness function, the prediction error of a candidate solution is used. In the figure, the true oscillator law and behavior are shown in dashed gray, a typical solution found by symbolic regression via GP is shown in solid black. It is important to note that GP did not only discover the solution structure, but also the constants defining the behavior of the concrete oscillator sampled.

Symbolic regression is not limited to numerical data. Another typical application is the induction of compact boolean expressions based on truth tables. Here, both independent and dependent variables are of boolean type.

The application domain for symbolic regression is very large, as the method can be used as an addition or a replacement for conventional regression methods. A non-exhaustive list of examples for concrete application fields include engineering, finance, and most fields of the natural and social sciences.[12]

[10] In the context of regression analysis, a *dependent variable* represents the output to be predicted by a model, also known as effect, or y variable. *Independent variables* represent the model inputs, also known as causes, or x variables.

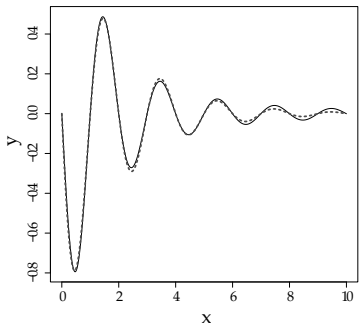


Figure 2.4: Symbolic regression result of the governing law of a damped oscillator. This example shows how symbolic regression via GP is used to rediscover the governing physical law of a damped oscillator. In contrast to many other regression methods, the solution is expressed as a mathematical formula accessible to human interpretation and validation. In this figure, the true oscillator behavior are shown in dashed gray, the (nearly exact) solution found by a GP run is shown in solid black.

[11] In this example, input data has been created by simulation based on known physical laws.

[12] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza)

2.2.2 *Classification*

Symbolic regression can be readily adapted for solving classification problems by using a single dependent variable of a type that ranges over the classes required. Features of the objects to be classified are then encoded as independent variables. Espejo et al. [2010] provide an extensive survey on classification via GP. [13] In addition to be applied directly to a classification task, GP can also be used as a component of more traditional classification approaches from machine learning and data mining.[14] For example, in the preprocessing phase, GP can be used for feature construction and feature selection. In the model extraction phase, GP can be used for evolving decision trees, classification rules, or discriminant functions. GP can be also used to evolve classifier ensembles. This is an example of the widespread approach of applying GP as a component of an existing problem specific methodology. Instead of exploring a more general but much larger search space, GP search is embedded into a well known solution framework, trading the potential for innovative high quality solutions for lower computational effort and solutions of predictable structure.

[13] Espejo, P. G., Ventura, S., and Herrera, F. (2010). A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(2):121–144

[14] Shearer, C. (2000). The CRISP-DM model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4):13–22

2.2.3 *Clustering*

GP has also been successfully applied to tasks in unsupervised learning, such as clustering.[15] Here, a data-driven GP system is configured with a fitness function that computes a quality measure for a given clustering. The concrete quality measure used is dependent on the application domain and on user preferences. In typical clustering applications, GP solutions are interpreted as functions taking an object feature vector as an input and returning a cluster label as an output. As the case in classification, GP can also be applied as a component of existing clustering algorithms, e.g., for data preprocessing or feature construction.

[15] Boric, N. and Estevez, P. A. (2007). Genetic programming-based clustering using an information theoretic fitness measure. In Srinivasan, D. and Wang, L., editors, *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 31–38, Piscataway, NJ. IEEE Press

2.2.4 *Automatic Programming*

As computer programs are symbolic expression, GP can of course be used for automatic programming, giving the method its name. This application requires a set of program building blocks and a fitness function that assigns a numerical fitness value to each candidate program. Depending on the set of building blocks and their syntactic rules of combination, i.e., the search space definition, the resulting language may or may not be Turing equivalent.[16] In practice, this approach is used successfully to evolve programs of small to medium scale and complexity in domain specific languages. Typical application areas include automatic programing of robot control programs, of programs that solve specific tasks in audio or video processing, or of programs that solve other control or signal processing tasks.

[16] Schöning, U. and Pruim, R. J. (1998). *Gems of Theoretical Computer Science*. Springer, Berlin

2.2.5 Other Applications

The applicability of GP goes beyond automatic programming. As mentioned earlier, the method can be used to discover all structures that are representable by symbolic expressions of moderate complexity. Examples include analog circuit design, antenna design, processing networks in manufacturing and logistics, strategies for algorithmic trading, or music composition and generative art. Willis et al. [1997] provides examples for additional applications.[17]

[17] Willis, M., Hiden, H., Marenbach, P., McKay, B., and Montague, G. A. (1997). Genetic programming: An introduction and survey of applications. In Zalzala, A., editor, *Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA 97)*, pages 314–319, London, U.K. IEE Conference Publications (No. 446)

2.3 A Short History of GP

At the time of this writing, studying GP in theory and practice has a tradition of nearly 30 years. Willis et al. [1997] gives a short but comprehensive overview of the early history of GP:

> Cramer [1985] developed one of the first tree structured genetic algorithms (GA) for basic symbolic regression. Another early development was the BEAGLE[18] algorithm of Forsyth [1986], which generated classification rules using a tree structured GA. However, it was Koza [1992, 1994] who was largely responsible for the popularisation of GP within the field of computer science. His GP algorithm (coded in LISP) was applied to a wide range of problems including symbolic regression, control, robotics, games and classification.

[18] Biological Evolutionary Algorithm Generating Logical Expressions

Following this early period, interest in the field of GP has steadily grown. The first international conference on the subject was held at Stanford University in 1996 (GP'96). While most research in the field was still done by computer scientists, first applications in engineering appeared. Most of these were introduced shortly in the previous section. A first generation of dedicated GP software systems were developed both in academics and commercially. The target audience for these systems were mostly researchers in GP or application domain specialists with strong backgrounds in computer science.[19]

[19] Examples of first-generation GP systems are TinyGP and Discipulus, see Section 3.1 for more details.

Recently, a second generation of commercial GP software was brought to market, which offers mature algorithms, good documentation, and user-friendly graphical interfaces.[20] For the first time, these systems enable users from many different application domains to experiment with GP.

[20] Section 3.1 contains detailed descriptions of the most important of these second generation systems.

2.4 Abstract Evolutionary Algorithms

This section introduces a basic formal framework for describing EAs in the abstract, i.e., without referring to a concrete implementation. This is motivated by three observations:

1. Formal description of EAs provides a framework for structuring a flexible and modular GP implementation.
2. Formal description of EAs provides the basis for formal definitions of all operators in a GP system, simplifying correct implementation.

3. Formal description of EAs provides an inroad to the developing mathematical theory of evolution.

Linking theory and implementation by basing both on the same formal grounding enables modules of the implementation to be adapted to expanding theoretical knowledge. Building an implementation on the most abstract domain concepts as efficiently possible with current technology constitutes good software engineering practice. Modularity and reusability are improved.

While the stance of this work is primarily empirical, given the fact that, at the time of writing, a mathematical theory of evolution in general and GP in particular is still in its infancy, theory provides an important complementary viewpoint. It already developed a useful formal language for describing the relation of search space geometry to the dynamics of evolutionary search.[21] GP schema theory, a branch of GP theory, developed exact models for predicting the dissemination of symbolic expressions fitting a given form or schema in the population during a GP run.[22] While empirical studies like the one presented in Chapter 5 can answer questions of what components of a GP system contribute to what degree to the system's performance, theory is beginning to offer answers to questions of why this is the case.

Today, much research in GP is of empirical nature, while a trend towards the use of solid statistical methods of experiment design and result analysis seems to be developing. This work pushes in the same direction by introducing new means of reproducible empirical research to the field of GP. The application of new theoretical results in the future, on the other hand, promises to improve GP system performance even further.

[21] Stadler, B. M. R., Stadler, P. F., Wagner, G. P., and Fontana, W. (2000). The topology of the possible: Formal spaces underlying patterns of evolutionary change. Working papers, Santa Fe Institute

[22] Poli, R. (2001). Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines*, 2(2):123–163

Evolutionary algorithms are heuristic population-based optimization algorithms.[23] Given a fitness function $\texttt{fit} :: P \rightarrow V$[24] mapping a solution $x \in P$ to a fitness value $\texttt{fit}(x) \in V$, where $(V, \sqsubseteq)$ is a partially ordered set, EAs are employed to find globally optimal solutions x^*. A globally optimal solution, or global optimum, x^* is a solution that satisfies the inequality

$$\texttt{fit}(x^*) \sqsubseteq \texttt{fit}(x) \textit{ for all } x \in P. \tag{2.1}$$

P is a possibly infinite set of valid candidate solutions, referred to as *phenotypes*, elements $v \in V$ are referred to as *fitness values* or objective function values.[25] In single-objective optimization, V is typically defined as the set of real numbers $\mathbb{R}$ and $\sqsubseteq$ is defined as the "less than or equal" relation on real numbers $\leq$. In this case, Equation 2.1 describes well-known single-objective fitness minimization.

In multi-objective optimization, V is typically defined as the n-dimensional vector space $\mathbb{R}^n$, where n is the number of optimization criteria. Here, $\sqsubseteq$ is typically defined via the Pareto dominance relation $\prec$.[26] The case of bi-objective optimization ($n = 2$) is particularly relevant to GP due to the necessity of controlling (minimiz-

[23] Bartz-Beielstein, T., Branke, J., Mehnen, J., and Mersmann, O. (2014). Evolutionary algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(3):178–195

[24] The :: operator means "is of type".

[25] In practice, Equation 2.1 is informally rephrased to "for most $x \in P$", i.e., an EA is run until a solution of satisfactory quality is found. This solution might be a local optimum instead of a global optimum.

[26] A vector x (Pareto-) dominates a vector x', written as $x \prec x'$, if each component of x is less than or equal to the corresponding component of x' and at least one component is strictly less, i.e., $\forall i : x_i \leq x'_i$ and $\exists j : x_j < x'_j$. This formulation leads to minimization of each vector component (objective).

ing) solution complexity in parallel to improving solution quality (minimizing an error measure).

For convenience and efficiency of implementation, some EAs and most GP algorithms do not operate on the application-dependent and possibly complex phenotype set P directly. Instead, a set G of *genotypes* is used as a proxy. An application-specific *genotype-phenotype mapping* $\mathtt{map} :: G \to P$ assigns a phenotype $\mathtt{map}(g) \in P$ to every genotype $g \in G$. A finite subset of G (P) of size μ is referred to as a genotypic (phenotypic) *population*, where $\mathtt{pop}_G(t) \in G^\mu$ ($\mathtt{pop}_P(t) \in P^\mu$) denotes a genotypic (phenotypic) population at discrete time t. The genotype-phenotype mapping $\mathtt{map}$ is trivially extended to genotypic populations by pointwise application: $\mathtt{map}_{\mathtt{pop}} :: 2^G \to 2^P$ where $\mathtt{map}_{\mathtt{pop}}(G) := \{\mathtt{map}(g)\}_{g \in G}$.[27] A single element of G (P) is referred to as an genotypic (phenotypic) *individual* or *candidate solution*.

[27] 2^X denotes the powerset of X, i.e., the set of all subsets of X.

Evolutionary algorithms use variation and selection operators to search the set of genotypes G while maintaining a genotypic population $\mathtt{pop}_G(t)$ at each point in time t. As EAs are randomized algorithms, both variation and selection operators can be probabilistic. A probabilistic influence on the behavior of these operators is modelled by tracking the state $\omega \in \Omega$ of an entropy source via an additional parameter and return value. Values of Ω represent the states of a random number generator in an implementation. Primitive functions are then used to extract usable random values from Ω.

The *random discrete uniform* primitive $r_{du} :: \Omega \times 2^X \to X \times \Omega$ selects a (uniformly) random element x from a set X based on entropy source state $\omega \in \Omega$. The *random normal* primitive $r_n :: \Omega \times \mathbb{R} \times \mathbb{R}_0^+ \to \mathbb{R} \times \Omega$, selects a random number $x = r_n(\omega, \mu, \sigma)$ from a normal distribution with mean μ and standard deviation σ. The *random uniform* primitive $r_u :: \Omega \times \mathbb{R} \times \mathbb{R} \to \mathbb{R} \times \Omega$ selects a random number $n = r_u(\omega, l, u)$ from an uniform distribution in the closed interval $[l, u] \subset \mathbb{R}$. In addition to x, these primitives also return a new entropy source state $\omega' \in \Omega$.

The notation $\mathtt{op} :: D \times \Omega \to R \times \Omega$ is used to describe a probabilistic operator $\mathtt{op}$ with domain D and range R, where additional parameters influencing the behavior of $\mathtt{op}$ are taken from D. This allows the use of "pure" functions to model probabilistic operators. In the remainder of this text, the "extra" parameter and return value from Ω is often omitted from formulas for notational brevity.

The most widespread evolutionary variation operators are recombination (also known as crossover) and mutation. The recombination operator $\mathtt{rec} :: G^\mu \to G^\lambda$ creates λ children from a parent generation of size μ. Practical recombination operators often only use a fixed-size subset of the parent generation to generate children. For example, binary crossover uses a subset of size 2. This subset is

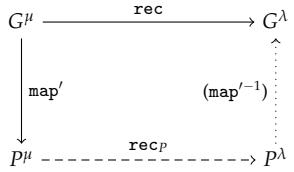


Figure 2.5: Domain and range of the genotypic recombination operator $\mathtt{rec}$ and construction of its phenotypic analogue $\mathtt{rec}_P$. A recombination operator is of type $\mathtt{rec} :: G^\mu \to G^\lambda$. In this diagram, $\mathtt{map}'$ denotes the pointwise extension of $\mathtt{map}$ to tuples. Note that the inverse genotype-phenotype mapping $\mathtt{map}'^{-1}$ is only defined if $\mathtt{map}'$ is injective.

selected randomly, or by means of a selection operator as described below. Well-behaved recombination operators that are useful in practice satisfy two basic axioms:

$$\{x, x'\} \in \mathcal{C}[\texttt{rec}](x, x'), \qquad \text{(reproduction)}$$
$$\mathcal{C}[\texttt{rec}](x, x') = \mathcal{C}[\texttt{rec}](x', x). \qquad \text{(commutativity)}$$

Here, $\mathcal{C}[\texttt{op}]$ is the *closure* of an operator `op`, which is defined as the family of all possible results of the application of a probabilistic operator `op`. The first axiom means that applying a recombination operator has a non-zero probability to return the parents unmodified, i.e., to reproduce them. The second axiom means that the order of the parents does not change the family of possible results of a recombination operator.

After the recombination operator has been applied, a mutation operator $\texttt{mut} :: G \to G$ is applied pointwise to each of the μ children: $\texttt{mut}_{\texttt{pop}} :: 2^G \to 2^G$ where $\texttt{mut}_{\texttt{pop}}(G) := \{\texttt{mut}(g)\}_{g \in G}$. Domain and range of recombination and mutation operators are shown in Figure 2.5 and Figure 2.6 respectively.

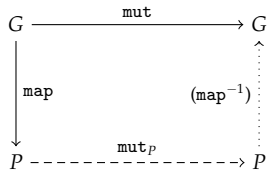


Figure 2.6: Domain and range of the genotypic mutation operator `mut` and construction of its phenotypic analogue $\texttt{mut}_P$. The inverse genotype-phenotype mapping $\texttt{map}^{-1}$ is only defined if `map` is injective.

A selection operator $\texttt{sel} :: G^\mu \times G^\lambda \to G^\mu$ is then applied to transfer the genotypic parent generation $\texttt{pop}_G(t)$ to the child generation $\texttt{pop}_G(t+1)$. In practice, a selection operator will often select the "best" λ individuals of the parent and children sets. A (μ, λ) selection strategy will only select individuals from the children created through recombination and mutation, whereas a $(\mu + \lambda)$ selection strategy will also consider unmodified parent individuals for selection.[28] Practical selection operators will base their decision on the single- or multi-objective fitness function `fit` and thus depend on `map`. Other selection criteria used in practice are purely genotypic properties such as complexity of the solution representation.

[28] By consequence, in a (μ, λ) strategy, the function that describes the population's best fitness over time t may not be monotonic.

A selection operator can also be applied when choosing parent individuals for recombination, as described above. This position of the selection operator, i.e., selection before recombination and mutation is known as *parent selection*, whereas selection after recombination and mutation is known as *survivor selection* or *environmental selection*. Practical algorithms mostly use survivor selection or survivor selection combined with parent selection, although all four combinations are possible.[29] The implementations of the survivor and parent selection used in an EA can differ, e.g., random parent selection combined with fitness proportional survivor selection.

[29] Although the fourth combination, i.e., using no selection at all, is irrelevant in practice.

Using the definitions above, a population at time $t+1$ can now be defined solely based on a population at time t, giving the general iteration scheme of an EA:

$$\texttt{pop}(t+1) := \texttt{sel}_{\text{survivor}}[\texttt{pop}(t), \texttt{mut}_{\texttt{pop}}(\texttt{rec}(\texttt{sel}_{\text{parent}}(\texttt{pop}(t))))] \tag{2.2}$$

There are multiple approaches of creating the *initial population* $\texttt{pop}(0)$, including random sampling from G or biased initialization

using solutions from a previous EA runs or solutions constructed by domain experts. There are also many possibilities in defining EA run stopping criteria. Usually the algorithm is run until a solution of sufficient quality is found or a preset compute time budget is exceeded.[30]

[30] More advanced stopping criteria involve online convergence detection and are often integrated with strategies for automatic restarts. See De Jong [2006] (pp. 78–79) for more details.

De Jong, K. A. (2006). *Evolutionary computation - A unified approach*. MIT Press, Cambridge, MA

The definitions above (Formula 2.2 and dependents) provide a general scheme for expressing abstract properties of EAs. In order to analyze performance characteristics of concrete EAs, this scheme needs to be realized by specific implementations of the genotype representation, the genotype-phenotype mapping, as well as the variation and selection operators. To associate abstract mathematical objects to implementations we use the syntactic convention $[\mathit{ObjectName}]\langle[\mathit{ImplementationName}]\rangle$. For example, a genotype set G realized by an n-dimensional vector space is written as $G\langle\mathbb{R}^n\rangle$ and the standard one-point crossover operator defined on this vector space as $\texttt{rec}\langle\texttt{ES-one-point}\rangle$. This convention makes it easy to link concrete implementations to the larger abstract framework of EAs.

The space of possible implementations of the iteration scheme given by Equation 2.2 exceeds the space of classical EAs. For example, it is possible to express random search, exhaustive search, and many other population-based search heuristics like particle swarm optimization, ant colony optimization, or estimation of distribution algorithms in terms of this scheme. Section 2.9 introduces search heuristics typically employed in GP and examined in this work.

2.4.1 *Abstract Genotypic and Phenotypic Search Spaces*

In the definitions above, both the set of possible genotypes G and the set of possible phenotypes P were just sets, without any additional structure. A wide array of methods available for analyzing and optimizing EA performance for specific application domains depends on the notion of genotypic and phenotypic *search space*, i.e., on the notion of adjacency, nearness, distance or accessibility in G and P.[31]

[31] Stadler, P. F. and Reidys, C. M. (2002). Neutrality in fitness landscapes. *Applied Mathematics and Computation*, 117(2–3):321–350

In EAs that use boolean or real vectors as genotypes (genetic algorithms and evolution strategies (ES)), the notion of a search space is highly intuitive as there are natural distance measures for boolean and real vector genotypes (Hamming distance and Euclidean distance). This intuition leads to the fact that a genotype set G is often implicitly identified with a *metric space* formed by (the carrier or support set) G and a natural distance measure d_G.[32] The distance measure d_G endows the genotype set G with a static structure independent of the fitness function f. Intuitively, d_G encodes how easy or probable or frequent it is to reach a genotype from another genotype during evolutionary search. In other words, the "nearness" of a genotype $g \in G$ to another genotype $g' \in G$ should approximate the probability of a sequence of variation (mutation and recombination) steps that turns g into g'.

[32] The natural distance measures being $d_G\langle\texttt{Hamming}\rangle$ in a GA and $d_G\langle\texttt{Euclidean}\rangle$ in an ES.

In EAs that use different genotypes, such as symbolic expressions represented by trees or graphs in GP, the natural choice for a genotype distance measure is less obvious. In combinatorial or discrete search spaces, distance measures that approximate how easily a genotype is reachable from another genotype are often not symmetric, leading to a notion of search spaces that is not a metric space. This is particularly true if non-homologous recombination operators are used, as commonly in GP. Stadler and Reidys [2002] describes how *pretopological spaces*, a generalization of the much more prevalent topological spaces can be employed to provide a theoretical basis for the analysis of these search spaces. A pretopological space consists of a set X and a collection $\mathcal{N}(x)$ of neighborhoods (a *neighborhood system* for every element $x \in X$ satisfying the following conditions:

$$
\begin{aligned}
N \in \mathcal{N}(x) &\implies x \in N, && \text{(centeredness)}\\
N \in \mathcal{N}(x) \text{ and } N \subseteq N' &\implies N' \in \mathcal{N}(x), && \text{(isotonicity)}\\
N, N' \in \mathcal{N}(x) &\implies N \cup N' \in \mathcal{N}(x). && \text{(directedness)}
\end{aligned}
$$

Concepts such as minima, maxima, or continuity of a function, connectedness, convergence, and limits can be defined on pretopological spaces. See Stadler and Reidys [2002] for details and a list of references. Formulating a mathematical theory of evolution in the language of generalized topology links the geometric properties of the search space as induced by the genotype set G, the variation operators and the genotype-phenotype mapping `map` to the dynamics of evolutionary search, such as convergence to a global optimum. In contrast to landscape analysis, this approach is able to describe important properties of these dynamics based on properties of the function `map`, without relying on the fitness function.[33] Being able to describe search space geometry and search dynamics without referring to a concrete fitness function also motivates the approach of tuning GP parameter settings for problem classes instead of for individual problem instances taken in Chapter 5.

[33] Stadler, B. M. R., Stadler, P. F., Wagner, G. P., and Fontana, W. (2000). The topology of the possible: Formal spaces underlying patterns of evolutionary change. Working papers, Santa Fe Institute

This work explicitly distinguishes between a genotype set G and a *genotypic search space* $\mathcal{G}$. The same distinction is made between a phenotype set P and *phenotypic search space* $\mathcal{P}$. A genotypic (phenotypic) search space is a pair consisting of G (resp. P) and a neighborhood system $\mathcal{N}_G$ ($\mathcal{N}_P$): $\mathcal{G} := (G, \mathcal{N}_G)$ (resp. $\mathcal{P} := (P, \mathcal{N}_P)$).

In the case of EAs with natural genotypic distance metrics, such as ES and GA, it is often simpler to define the search space by means of these metrics. Among other things, this allows to study causality in evolutionary search.[34] A metric is a function $d :: X \times X \to \mathbb{R}$ that satisfies the following conditions for each

[34] Droste, S., Jansen, T., and Wegener, I. (2000). Optimization with randomized search heuristics: The (A)NFL theorem, realistic scenarios, and difficult functions. Interner Bericht des Sonderforschungsbereichs 531 *Computational Intelligence* CI–91/00, Universität Dortmund, Germany

$a, b, c \in X$:

$$
\begin{aligned}
&d(a,b) \geq 0, && \text{(non-negativity)}\\
&d(a,b) = 0 \iff a = b, && \text{(identity of indiscernibles)}\\
&d(a,b) = d(b,a), \text{ and} && \text{(symmetry)}\\
&d(a,c) \leq d(a,b) + d(b,c). && \text{(triangle inequality)}
\end{aligned}
$$

Relying on this definition, the neighborhood structure of X can be defined via the concept of an *open ball*: Given a metric d defined on X and an element $x \in X$, the open ball $B_\epsilon(x)$ around x of radius ϵ is defined as the set

$$B_\epsilon(x) := \{x' \in X | d(x, x') < \epsilon\}. \tag{2.3}$$

The open ball around x or radius ϵ is known as an ϵ-neighbourhood. It contains all elements $x' \in X$ whose distance d from x is strictly less than ϵ. All open balls around x then constitute the neighborhood system $\mathcal{N}(x)$.

As mentioned above, the abstract definition of genotypic and phenotypic search spaces enables a wide array of methods for analyzing and optimizing EA performance. Examples include landscape analysis[35], search space visualization[36], fitness distance correlation analysis[37], or locality analysis of the genotype-phenotype mapping[38]. Genotypic and phenotypic search spaces formalize the intuition of individual and solution similarity. In the following sections, concrete examples of genotype and phenotypes representations for GP, as well as genotypic and phenotypic search spaces for GP, will be given. This work focuses on GP, nevertheless it would be equally possible to define other EAs in this framework by supplying suitable representations and distance measures.

[35] Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., and Rudolph, G. (2011). Exploratory landscape analysis. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation (GECCO 2011)*, pages 829–836, New York. ACM Press

[36] Pohlheim, H. (2006). Multidimensional scaling for evolutionary algorithms - Visualization of the path through search space and solution space using sammon mapping. *Artificial Life*, 12(2):203–209

[37] Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA. Morgan Kaufmann

[38] Raidl, G. R. and Gottlieb, J. (2005). Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4):441–475

2.5 Genotypes, Phenotypes and Fitness in GP

Based on the abstract definitions of genotype and phenotype given in Section 2.4, this section provides definitions of genotypes and phenotypes used in GP. It should become clear in this section that it is possible to define GP genotypes without resorting to concrete genotypic representations like trees, graphs, or register machine programs. That does not mean that concrete representations are of no importance. As mentioned in the introduction and explained in more detail in the next subsection, concrete representations and variation operators play a crucial role in GP algorithm performance. By defining GP genotypes as expressions of a formal language, it is possible to give precise formal specifications of many important operations on these genotypes, including initialization and variation operators and genotypic (distance) metrics.

2.5.1 GP Genotypes

The set of S-expressions[39] typically used for individual represen-

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

tation in (tree) GP is isomorphic to the set of terms in first-order logic. This syntactic equivalence motivates the following definition of the set of abstract GP genotypes: Given a set of input variable symbols V and a family of function symbols F, where F_i is the set of function symbols of arity i, the set of abstract GP genotypes G_{GP} can be defined inductively:

$$V \subseteq G_{GP}, \qquad \text{(Input Variables)}$$
$$f(t_1, \ldots, t_n) \in G_{GP} \text{ if } f \in F_n \text{ and } \{t_1, \ldots, t_n\} \in G_{GP}. \qquad \text{(Function Applications)}$$

F_0 is also known as the set of constant symbols, or simply *constant set*. The disjoint union over all sets in the F_n, with $n > 0$, is commonly known as the *function set*. In the reminder of this text, elements of the set of abstract GP genotypes will simply be referred to as GP *expressions*. Again, it is important to note that the set of abstract GP genotypes defined above is isomorphic to the set of S-expression commonly used in tree GP, meaning that there is a one to one mapping from S-expressions to abstract GP genotypes. It is therefore possible to describe complex operations less formally in pseudo-code or via pictures of expression trees. This text employs formal and informal descriptions interchangeably or in parallel, depending on what is more appropriate and clear in a given situation.

To arrive at concrete GP genotypes, concrete instances of V and F, as well as a concrete representation (trees, graphs, register machine programs, etc.) must be provided. Concrete representations primarily influence the time, memory, and implementation complexity of operations on genotypes, all of which are important topics but lie outside the scope of this chapter. The well-known test problems *artificial ant* and *simple symbolic regression*[40] provide concrete examples for V and F.

For the artificial ant problem, the combined set F consists of the sets $F_0 := \{\texttt{left}, \texttt{right}, \texttt{move}\}$, $F_2 := \{\texttt{if\_food\_ahead}, \texttt{progn2}\}$ and $F_3 := \{\texttt{progn3}\}$. In this problem, the input variable set V is empty, as solution programs receive input implicitly. An example for a GP genotype in this problem would be `if_food_ahead(progn2(move, left), right)`.

For the simple (univariate) symbolic regression problem, the input variable set is defined as $V := \{\texttt{x}\}$. By extending the input variable set to $V := \{\texttt{x}_1, \ldots, \texttt{x}_\texttt{d}\}$, this definition can readily be extended to multivariate symbolic regression for d-dimensional input data. The family of function sets F consists of $F_1 := \{\texttt{sin}, \texttt{cos}, \texttt{exp}, \texttt{log}\}$ and $F_2 := \{+, -, *, /\}$. In Koza [1992]'s formulation[41], no constant set is defined, but could be easily added by including a set $F_0 := \texttt{R}$ into F, where $\texttt{R}$ is a finite subset of $\mathbb{R}$. An example genotype for this multivariate symbolic regression with constants problem could be $+(\texttt{sin}(\texttt{x}_1), *(2.14, \texttt{x}_2))$, which in a more familiar infix syntax would be written as $\texttt{sin}(\texttt{x}_1) + 2.14 * \texttt{x}_2$.

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

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

2.5.2 GP Phenotypes

In GP, the genotype-phenotype mapping $\texttt{map}_{GP} :: G_{GP} \rightarrow P_{GP}$ is defined by giving a formal denotational semantics for the abstract GP genotypes defined in Subsection2.5.1. The specifics of these semantics depend on the semantics of the function symbols contained in F. As an example, the semantics for the univariate symbolic regression problem without constants $SR1$, $\texttt{map}_{GP-SR1} :: G_{GP-SR1} \rightarrow (\mathbb{R} \rightarrow \mathbb{R})$ are presented below:

$$
\begin{aligned}
\texttt{map}_{GP-SR1}[t] &:= f(x) \rightarrow \mathcal{E}(t), \text{where} \\
\mathcal{E}[\texttt{x}] &:= x, && \text{(Input Variable)} \\
\mathcal{E}[\texttt{sin}(t)] &:= \sin(\mathcal{E}[t]), && \text{(Sine)} \\
\mathcal{E}[\texttt{cos}(t)] &:= \cos(\mathcal{E}[t]), && \text{(Cosine)} \\
\mathcal{E}[\texttt{exp}(t)] &:= \exp(\mathcal{E}[t]), && \text{(Exponential)} \\
\mathcal{E}[\texttt{log}(t)] &:= \log(\mathcal{E}[t]), && \text{(Natural Logarithm)} \\
\mathcal{E}[t_1 + t_2] &:= \mathcal{E}[t_1] + \mathcal{E}[t_2], && \text{(Addition)} \\
\mathcal{E}[t_1 - t_2] &:= \mathcal{E}[t_1] - \mathcal{E}[t_2], && \text{(Subtraction)} \\
\mathcal{E}[t_1 * t_2] &:= \mathcal{E}[t_1] \times \mathcal{E}[t_2], && \text{(Multiplication)} \\
\mathcal{E}[t_1 / t_2] &:= \mathcal{E}[t_1] \div \mathcal{E}[t_2]. && \text{(Division)}
\end{aligned}
$$

Here, $\mathcal{E}$ is the denotational semantics of input variables and function applications. These semantics map an abstract GP genotype t, i.e., a symbolic expression, to a function with domain $\mathbb{R}$ and range $\mathbb{R}$. For example, in this semantics, the abstract GP genotype $\texttt{sin}(\texttt{x} * \texttt{x})$ denotes the function $f(x) \rightarrow \sin(x^2)$.

The definition of a GP phenotype is dependent on the application, e.g., symbolic regression, but independent of the concrete problem instance, e.g., symbolic regression of a damped oscillator. It offers a middle-ground for analysing the performance of GP algorithm components on specific applications, independent of concrete problem instances. A concrete fitness function F, on the other hand, is both application dependent and problem dependent.

2.5.3 GP Fitness Functions

The fitness function defines the interface between a solution space P and selection operator `sel` and as thus guides the GP search process to desirable solutions. In the case of GP, desirable solutions are 1) accurate, 2) compact, 3) understandable and 4) generalizable.

In this enumeration, accuracy is highly problem dependent. For example, an accurate solution for a classification problem is a classifier that provides high precision and recall, often combined into measures like the F-score[42] or Cohen's Kappa coefficient[43].

Solution compactness on the other hand, can be measured independently of the concrete problem, for example by statistical information criteria or, more prevalently, by syntactical solution

[42] Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann, Newton, MA, 2nd edition

[43] Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37

size and complexity measures. The importance of regarding solution compactness as a contributing factor to overall solution quality or fitness is amplified by the phenomenon of bloat in GP, i.e., the tendency of the average solution size to grow during evolution.[44]

[44] Silva, S. (2008). *Controlling bloat: Individual and population based approaches in genetic programming*. PhD thesis, Departamento de Engenharia Informatica, Universidade de Coimbra, Portugal

Understandability cannot easily defined mathematically as it depends on prior knowledge on the side of the GP system user. Nonetheless, the understandability of a solution is certainly correlated with its complexity. Other variables influencing understandability are the solution dimensionality, in applications where the concept applies, the number and the kind of functions used from the function set, as well as the similarity to solutions known in the application field.

Solution generalizability can be estimated by validation on an independent test cases or, in the case of machine learning problems, by more advanced methods such as cross-validation. In practice, solutions of lower complexity often achieve higher generalizability, but this is not always the case in all applications (cf. Vladislavleva et al. [2009]). In the important application of symbolic regression, more explicit generalizability measures have been recently developed.[45] As the phenomenon of bloat amplifies the importance of solution compactness, the importance of solution generalizability is amplified by the phenomenon of overfitting.

[45] Vladislavleva, E., Smits, G. F., and Den Hertog, D. (2009). Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349

Multiple solutions were proposed to handle the inherently multi-objective nature of GP fitness. The simplest solution is to combine objectives like accuracy and compactness into a scalar fitness value, e.g., via a weighted sum. There are many issues to this solution, though. It is often not clear how to weight individual objectives and objectives are often defined on largely different scales. Furthermore, Pareto-optimal solutions might become unreachable.

In modern GP systems, fitness is defined as a vector of real numbers, i.e., $\texttt{fit}_{GP} :: P \rightarrow \mathbb{R}^n$, where n is the number of objectives. In common practice at least two objectives are used, the first objective being solution accuracy, the second being solution compactness. These fitness vectors are then consumed by a multi-objective selection operator as detailed in Section 2.6.4.

Concrete measures for solution compactness are intuitively defined on abstract GP genotypes. A non-exhaustive enumeration of these measures include 1) the number of function symbols, constants and input variables in a symbolic expression, 2) the number of levels in the tree representation of a symbolic expression and 3) the internal or external path length of the tree representation of a symbolic expressions. All of these measures are used in practice. A problem with these measures is their coarse granularity, i.e., in each of these measures, many symbolic expressions are mapped to the same complexity value. To alleviate this and other problems, Keijzer and Foster [2007] introduce the measure of expression *visitation length* $\texttt{vl} :: G_{GP} \rightarrow \mathbb{N}$, which is defined in Equation 2.4,

where $c(t)$ denotes the number of children of the root of the tree representation of t and t_i denotes the ith subtree of t.[46]

$$\texttt{vl}(t) := s(t) + \sum_{i=1}^{c(t)} \texttt{vl}(t_i) \tag{2.4}$$

For symbolic expressions t of fixed size $s(t)$, $\texttt{vl}(t)$ will give smaller values values for balanced trees, and larger values for unbalanced trees. It can be used to steer GP search towards small, balanced solution expressions.

[46] Keijzer, M. and Foster, J. (2007). Crossover bias in genetic programming. In Ebner, M. et al., editors, *Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 33–44. Springer, Berlin

The focus of this work lies on data-driven GP and on symbolic regression in particular. Here, solution accuracy can be defined via well-known error measures, i.e., measures of the difference between observed values $\hat{y}$ and values y predicted by a model. Error measures commonly used in GP are mean-absolute error (MAE, Equation 2.5), sum-square error (SSE, Equation 2.6), mean-square error (MSE, Equation 2.7), and root-mean-square error (RMSE, Equation 2.8).

$$\texttt{mae}(\hat{y}, y) := \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i| \tag{2.5}$$

$$\texttt{sse}(\hat{y}, y) := \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{2.6}$$

$$\texttt{mse}(\hat{y}, y) := \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{2.7}$$

$$\texttt{rmse}(\hat{y}, y) := \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2} \tag{2.8}$$

The MAE measures the average magnitude of the errors of a model in relation to observed values. It uses the same units as the original data and is easy to understand and efficiently implementable. It is linear in the sense that all individual differences are weighted equally during the calculation of the mean. Willmott and Matsuura [2005] shows advantages of the MAE compared to the more prevalent RMSE when used for model comparison.[47]

[47] Willmott, C. J. and Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30(1):79–82

The SSE measures the sum of squares of the errors of a model in relation to observed values. It is non-linear in the sense that it gives higher weight to large individual differences. It is measured in squared units, making it harder to interpret than linear measures, but more efficient than non-linear measures that use the same units as the original data.

The RMSE measures the average magnitude of the errors of a model in relation to observed values. It is non-linear, giving higher weight to large individual differences, but uses the same units as the original data. The RMSE is influenced not only by the average error magnitude (MAE), but also by the error variance, which can introduce unwanted confounding in model comparisons. As the SSE, the RMSE is most useful when models with occasional high

individual errors are particularly undesirable. Its prevalence makes it useful for comparing results of different models, although the MAE should be preferred for this purpose, due to the confounding issues mentioned above.

RMSE and MAE can be used to estimate the variation of the errors of a model. The RMSE is always larger than the MAE value, where a large difference denotes high variance of the individual differences. The image of MAE, SSE and RMSE is the set of positive real numbers including zero $\mathbb{R}_0^+$, where lower values are better.

$$\texttt{smse}(\hat{y}, y) := \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - (a + by_i))^2 \tag{2.9}$$

Some modern symbolic regression systems implement a linear regression step on the output of symbolic expressions, freeing GP search from discovering scaling constants and thus improving search efficiency. This leads to the scaled-mean-square error measure (SMSE, Equation 2.9), where scaling constants a and b are calculated via a least-squares fit. Keijzer [2004] proves that SMSE is expected to perform better than unscaled error measures on all possible symbolic regression problems.[48] SMSE puts the upper bound $\mathrm{Var}(\hat{y})$ on the error, simplifying selection operators as well as result comparison and visualization.

[48] Keijzer, M. (2004). Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):59–269

It is important to note that all of the measures described above cannot be compared across different data sets as they are scale dependent. Hyndman [2006] describes scale free alternatives that are not common in the field of GP though, as they are slightly less efficient to calculate.[49] The same applies to the (scale free) coefficient of determination R^2, which is sometimes used in other literature when comparing the accuracy of GP models to other modeling approaches across data sets. The R^2 measure is related to the SSE by the equation $R^2 = 1 - \texttt{sse}/\texttt{tsse}$, where TSSE denotes the total sum of squares, which is proportional to the sample variance. Therefore, R^2 is related to the unexplained variance of a model prediction. Note that R^2 is only defined for linear models, and becomes meaningless if the data to be modeled is strongly non-linear.

[49] Hyndman, R. J. (2006). Another look at forecast accuracy metrics for intermittent demand. *Foresight: The International Journal of Applied Forecasting*, 2006(4):43–46

2.6 GP Search Operators

GP search operators, i.e., operators for population initialization, variation and selection, are the means by which a GP system navigates the set of possible genotypic solutions. Via the genotype-phenotype mapping $\texttt{map}_{GP}$, these operators also define how the set of phenotypic solutions P is navigated. As concrete implementations of most of these operators strongly depend on the GP representation used,[50] each GP representation tends to use its own set of initialization and variation operators.[51]

[50] The exception here being selection, which is representation-agnostic.

[51] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza)

The notion of abstract GP genotypes as introduced in Section 2.5.1 enables the formal definition of GP search operators without resorting to a concrete representation. In turn, this en-

ables the independent study of the influence of GP search operators and GP representations on GP performance. For many practical application problems, the most widespread GP representations are semantically equivalent, but gravitate towards different sets of GP search operators, depending on which set of operators has the most efficient implementation on a certain representation.

Traditional GP search operators manipulate symbolic expressions regardless of their semantics as given by a genotype-phenotype mapping $\texttt{map}_{GP}$. Recently, semantically aware search operators have been introduced and shown to outperform purely syntactic operators in certain problem domains.[52] These operators have to incorporate background knowledge on the structure of a concrete genotype-phenotype mapping $\texttt{map}_{GP}$ and are therefore dependent on the problem class. At the time of writing, existing examples of semantically aware GP search operators for symbolic regression also suffer from the need to grow genotypes in size with each variation step, leading to very complex solutions that, while providing high accuracy, suffer from low generalizability and low understandability.

[52] Moraglio, A., Krawiec, K., and Johnson, C. G. (2012). Geometric semantic genetic programming. In *Parallel Problem Solving from Nature-PPSN XII*, pages 21–31. Springer, Berlin

This work focuses on the well-known traditional GP search operators originating in tree-based GP as popularized by Koza [1992]. In the following Subsections 2.6.1 – 2.6.4, formal definitions of the operators for initialization, variation and selection are presented. These operators are then employed in the GP systems studied empirically in later chapters.

2.6.1 *Initialization Operators*

Initialization operators are used to create the initial population of symbolic expressions. They are also used to create new subexpressions in certain variation operators, as well as, in certain search heuristics, to generate new solutions to insert into a population during GP search as a means of diversity preservation.

With conventional GP search heuristics at least, the composition of the initial population has a significant impact on the performance of a GP run. As the number of possible symbolic expressions is infinite, there is no bias-free initialization strategy.

Random Initialization Below, Equation 2.10 defines the *random initialization* operator $\texttt{init} :: \mathbb{N} \times \mathbb{P}^2 \rightarrow G_{GP}$. Here, $\mathbb{P} := [0, 1]$ denotes the set of probabilities, i.e., real numbers in the closed interval between zero and one. The set of functions with arity equal

to or greater than one is denoted by $F_{>0} := \bigcup_{i>0} F_i$.

$$\begin{aligned} \texttt{init}(n, p_s, p_v) &:= \mathcal{I}(0, n, p_s, p_v) \\ \mathcal{I}(i, n, p_s, p_v) &:= \begin{cases} f(\underbrace{\mathcal{I}(i+1, n, p_s, p_v), \ldots, \mathcal{I}(i+1, n, p_s, p_v)}_{\texttt{arity}(f)}) & \text{if } r_u(0,1) < p_s \text{ and } i < n \\ r_{du}(V) & \text{if } r_u(0,1) < p_v \\ r_{du}(F_0) & \text{otherwise} \end{cases} \\ f &:= r_{du}(F_{>0}) \end{aligned} \tag{2.10}$$

Random initialization creates symbolic expressions of maximum depth n. In each recursion step, a subexpression is created with probability p_s. If no subtree is created, an input variable is created with probability p_v. Otherwise, a constant is created. By setting $p_s := 1$ to one and $p_v := {}^{|V|}/_{(|V|+|F_0|)}$ according to the ratio between the number of input variables and the number of input variables and constants (the number of terminals), only symbolic expressions represented by full trees of depth n are created, realizing Koza [1992]'s *full* initialization strategy. Koza [1992]'s *grow* initialization strategy can be realized by setting p_v as in the full strategy and $p_s := {}^{|F_{>0}|}/_{|V|+|F|}$ according to the ratio of the number of functions of arity equal to or greater than one and the number of all functions and input variables.

As neither the grow or full strategies create a very wide array of sizes or shapes, Koza [1992] introduced the *ramped half-and-half* strategy combining both methods. One half of the initial population is created using the full strategy, while the other half is created with the grow strategy. Both strategies are applied using a range of depth limits to ensure high variability in expression size and shape.

Other Initialization Operators With the initialization operators introduced above, the distribution of sizes and shapes of the symbolic expressions generated is dependent on the sets F and V in non-trivial ways that can be studied with methods from analytic combinatorics[53]. This leads to initialization strategies that offer tighter control over the statistical properties of the symbolic expressions created.[54]

Of course, it is also possible to start a GP run from a set of known solutions represented as symbolic expressions. This approach is known as *seeding*. Poli et al. [2008] describe different variants and best practices regarding this approach.

[53] Flajolet, P. and Sedgewick, R. (2009). *Analytic Combinatorics*. Cambridge University Press, New York, 1st edition

[54] Bohm, W. and Geyer-Schulz, A. (1996). Exact uniform initialization for genetic programming. In Belew, R. K. and Vose, M., editors, *Foundations of Genetic Algorithms IV*, pages 379–407, San Francisco, CA. Morgan Kaufmann; and Iba, H. (1996). Random tree generation for genetic programming. In Voigt, H. et al., editors, *Parallel Problem Solving from Nature IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 144–153, Berlin. Springer

2.6.2 *Mutation Operators*

Mutation operators modify a single symbolic expression. While early approaches of program synthesis through EAs employed mutation operators[55], the first GP systems as popularized by Koza [1992] did not and relied on recombination exclusively.

Wether mutation is beneficial or detrimental to GP system performance depends on the combination of the concrete mutation

[55] Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In Grefenstette, J. J., editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, New York. Psychology Press

operator used and the problem class.[5][6] In symbolic regression, a mutation operator is often used to enable the adaption of constants during evolutionary search.

This section defines the mutation operators used in the GP system studied in this work. The focus lies on operators common both in the literature and in practical GP systems. In addition to giving abstract definitions, practical examples of the effects of mutation on genotype and phenotype in the problem class of symbolic regression are provided.

[5][6] Luke, S. and Spector, L. (1997). A comparison of crossover and mutation in genetic programming. In Koza, J. R. et al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 240–248, San Francisco, CA. Morgan Kaufmann

Label Mutation Below, Equation 2.11 defines the *label mutation* operator $\texttt{mut}_l :: G_{GP} \times \mathbb{P} \rightarrow G_{GP}$. In this definition $v \in V$ denotes an input variable.

$$\begin{aligned}
\texttt{mut}_l(f(t_1,\ldots,t_n),p) &:= \begin{cases} f'(\texttt{mut}_l(t_1,p),\ldots,\texttt{mut}_l(t_n,p)) & \text{if } r_u(0,1) < p \\ f(\texttt{mut}_l(t_1,p),\ldots,\texttt{mut}_l(t_n,p)) & \text{otherwise} \end{cases} \\
\texttt{mut}_l(v,p) &:= \begin{cases} r_{du}(V) & \text{if } r_u(0,1) < p \\ v & \text{otherwise} \end{cases} \\
f' &:= r_{du}(F_{\texttt{arity}(f)})
\end{aligned} \tag{2.11}$$

Label mutation modifies a symbolic expression by randomly replacing function symbols with function symbols of the same arity, and by replacing input variables by input variables. The strength of the operator is controlled by the parameter p, denoting the probability to replace the current root node label while the operator is recursively applied to each subexpression of a symbolic expression. Note that the shape of the tree representation of the symbolic expression is left unchanged by label mutation.

Figure 2.7 shows an example for the effect of label mutation on genotype and phenotype in symbolic regression of unary functions. The upper panels show genotype and phenotype before the operator has been applied, the lower panels show genotype and phenotype after operator application. Affected tree nodes are highlighted by a wide gray border. Operator strength was set to a low value of $p := 0.05$, and only one node label was affected accordingly. Nonetheless, phenotypic behavior is changed considerably.

Subtree Mutation Below, Equation 2.12 defines the *subtree mutation* operator $\texttt{mut}_s :: G_{GP} \times \mathbb{N} \times \mathbb{P}^4 \rightarrow G_{GP}$.

$$\begin{aligned}
\texttt{mut}_s(f(t_1,\ldots,t_n),n,p,p_i,p_s,p_v) &:= \begin{cases} \texttt{init}(n,p_s,p_v) & \text{if } r_u(0,1) < p \text{ and } r_u(0,1) < p_i \\ \texttt{init}(0,p_s,p_v) & \text{if } r_u(0,1) < p \text{ and } r_u(0,1) \geq p_i \\ f(\texttt{mut}_s(t_1,n,p,p_i,p_s,p_v),\ldots, & \\ \quad \texttt{mut}_s(t_n,n,p,p_i,p_s,p_v)) & \text{otherwise} \end{cases} \\
\texttt{mut}_s(v,n,p,p_i,p_s,p_v) &:= \begin{cases} \texttt{init}(n,p_s,p_v) & \text{if } r_u(0,1) < p \text{ and } r_u(0,1) < p_i \\ \texttt{init}(0,p_s,p_v) & \text{if } r_u(0,1) < p \text{ and } r_u(0,1) \geq p_i \\ v & \text{otherwise} \end{cases}
\end{aligned} \tag{2.12}$$

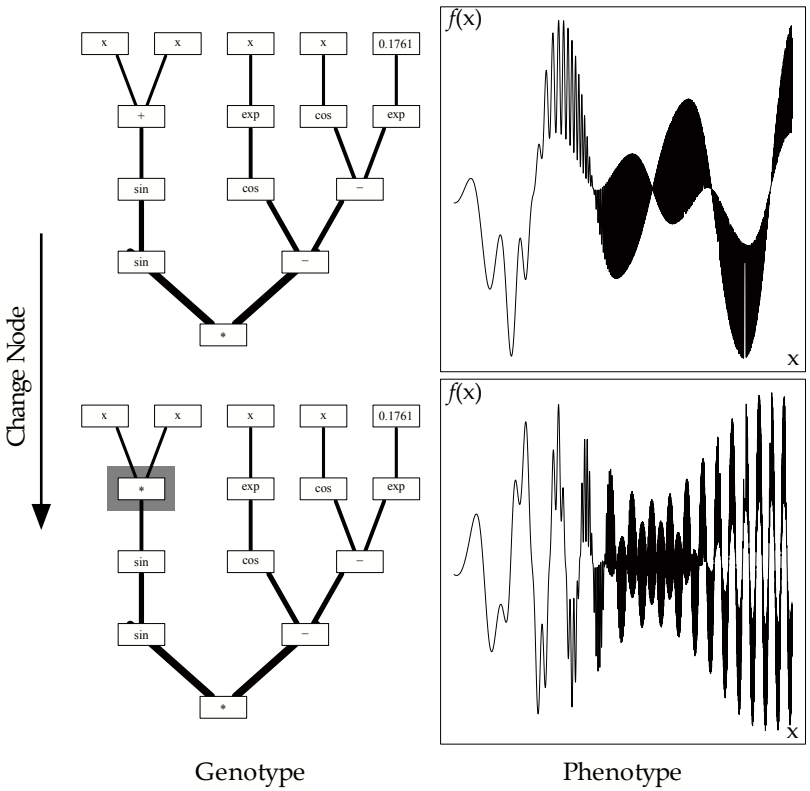


Figure 2.7: Example of the effect of the label mutation operator on genotype and phenotype in symbolic regression of unary functions. The upper panels show genotype and phenotype before the operator has been applied, the lower panels show genotype and phenotype after operator application. Affected tree nodes are highlighted by a wide gray border. Operator strength was set to a low value of $p := 0.05$, therefore only one node label was affected. Nonetheless, the behavior of the phenotype is changed considerably.

Subtree mutation modifies a symbolic expression by randomly replacing subexpressions by newly initialized expressions or by input variables or constants. The maximum size, as well as the shape of the newly initialized subexpressions can be controlled by the parameters n, p_s, and p_v, with the same semantics as in Equation 2.10.

Operator strength can be controlled by the parameter p, denoting the probability to replace a subexpression with a newly initialized expression while the operator is recursively applied to each subexpression. The parameter p_i denotes the probability of a newly created expression to be a constant or input variable, providing control over the tendency of the operator to grow or shrink expressions.

Figure 2.8 provides an example of the genotypic and phenotypic effect of subtree mutation in the context of symbolic regression of unary functions. The same format and example expression as in Figure 2.7 is used. In this example, an input variable is replaced by a newly initialized subexpression of three nodes, increasing the overall number of nodes by two and the total depth of the expression by one.

Figure 2.9 provides an example of subtree mutation analogous to the one shown in Figure 2.8. This time, a subexpression of three nodes is replaced by an input variable, reducing the overall number of nodes by two. These examples illustrate that in symbolic regression, there is no easily discernible relation between genotypic expression size and (apparent) phenotypic behavior complexity.

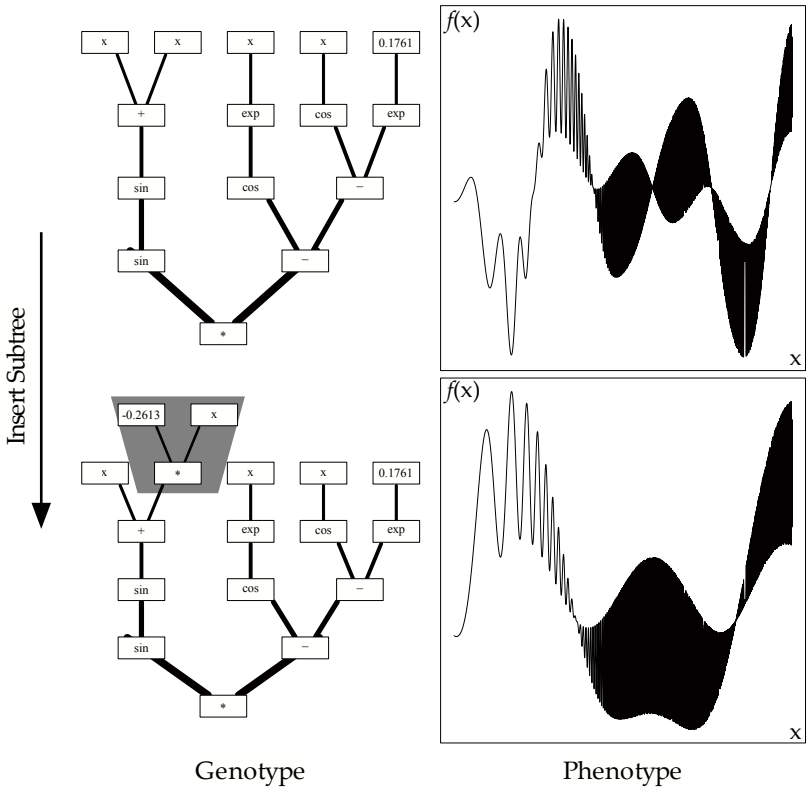


Figure 2.8: Example of the genotypic and phenotypic effect of subtree mutation in the context of symbolic regression of unary functions. The same format and example expression as in Figure 2.7 is used. In this example, an input variable is replaced by a newly initialized subexpression of three nodes, increasing the overall number of nodes by two and the total depth of the expression by one.

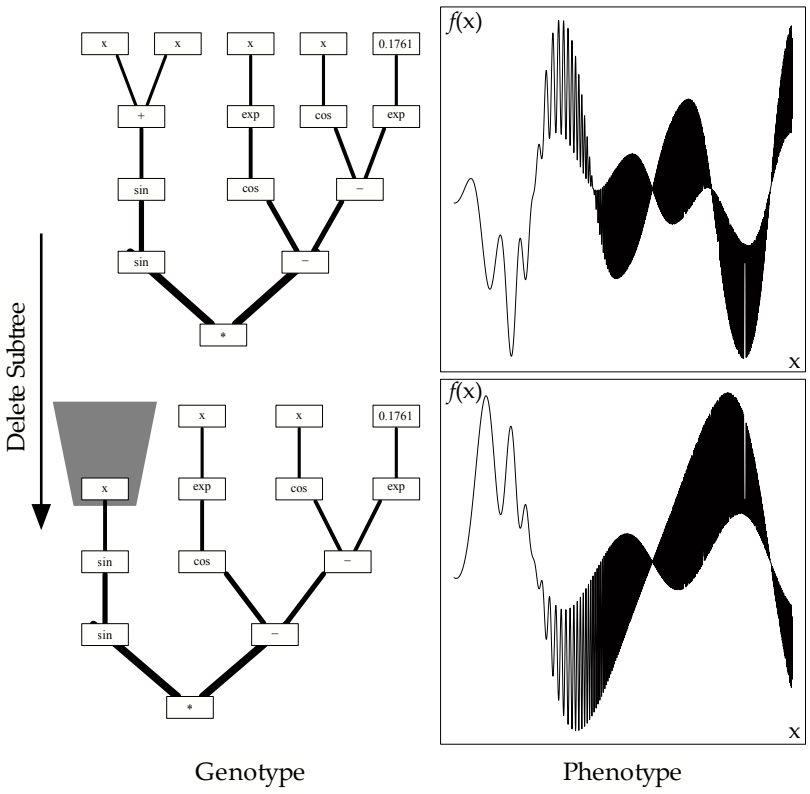


Figure 2.9: Example of subtree mutation analogous to the one shown in Figure 2.8. Here, a subexpression of three nodes is replaced by an input variable, reducing the overall number of nodes by two.

Numeric Constant Mutation In GP application classes such as symbolic regression, the selection of values for constants in symbolic expressions is required. A common approach is to use the same evolutionary search process to discover these constants that is used for discovering the solution expression structure. Equation 2.13 defines the *numeric constant mutation* operator $\texttt{mut}_c :: G_{GP} \times \mathbb{R} \times \mathbb{R}_0^+ \times \mathbb{P} \rightarrow G_{GP}$. This operator assumes the presence of numeric constants $\mathbb{R} \subseteq F_0$.

$$\begin{aligned} \texttt{mut}_c(f(t_1,\ldots,t_n),p,\mu,\sigma) &:= \begin{cases} f + r_n(\mu,\sigma) & \text{if } r_u(0,1) < p \text{ and } f \in \mathbb{R} \\ f(\texttt{mut}_c(t_1,p,\mu,\sigma),\ldots,\texttt{mut}_c(t_n,p,\mu,\sigma)) & \text{otherwise} \end{cases} \\ \texttt{mut}_c(v,p,\mu,\sigma) &:= v \end{aligned} \tag{2.13}$$

Here, $f \in \mathbb{R}$ denotes a real-valued function of arity 0, i.e., a constant. As in the mutation operators presented earlier, operator strength can be controlled by the parameter p, denoting the probability to mutate each numeric constant present in a symbolic expression. Numeric constants are mutated by adding a random number drawn from a normal distribution with mean μ and standard deviation σ. Mean and standard deviation are parameters to the operator.

Other Mutation Operators As the case with GP initialization operators, there are many different GP mutation operators described in the literature. Piszcz and Soule [2006b] provide a comprehensive and systematic survey of this topic.[57]

There are at least three reasons for the diversity in GP mutation operators. First, different GP representations lend themselves to different sets of easily and efficiently implementable mutation operators. Second, fine tuning mutation operators to certain application classes is common in GP. The numeric constant mutation operator introduced above is a basic example for this practice. Third, lack of pervasive knowledge of GP theory and the observation that search operators are seldom defined formally in GP literature leads to the repeated reinvention of mutation operators in different GP systems.

While the GP community today judges the performance of different GP mutation operators mostly by referring to experimental results, there are some guiding principles to lead GP users and GP system implementers in the selection of a "good" operator set.

Droste and Wiesmann [2000] introduce the framework of metric-based EAs, which contains testable properties of mutation and recombination operators with provably favorable performance characteristics.[58] Unfortunately, the complex nature of the genotype-phenotype mapping in many GP application classes, and that includes the important class of symbolic regression, makes it extremely difficult to design and implement operators that fulfill these properties without losing the important benefit of compact and understandable solutions. On the other hand, the metric-based approach to GP operator design proved very valuable on applica-

[57] Piszcz, A. and Soule, T. (2006b). A survey of mutation techniques in genetic programming. In Keijzer, M. et al., editors, *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO 2006)*, volume 1, pages 951–952, Seattle, WA. ACM Press

[58] Droste, S. and Wiesmann, D. (2000). Metric-based evolutionary algorithms. In *Proceedings of the European Conference on Genetic Programming (EuroGP 2000)*, volume 1802 of *Lecture Notes in Computer Science*, pages 29–43, Berlin. Springer

tion classes with less complex genotype-phenotype mappings. An important example is the data-driven synthesis of boolean functions represented as ordered binary decision diagrams.

2.6.3 *Recombination Operators*

GP recombination operators, also called crossover operators, recombine multiple symbolic expressions. In practice, common GP recombination operators combine two parent expressions to form one or two child expressions. GP recombination operators are used with the intuition that subexpressions of high fitness solutions can evolve independently. In many practical GP application areas, this requirement is at least fulfilled to some degree.

Standard Crossover Equation 2.14 defines the *standard crossover* operator $\texttt{rec}_{GP} :: G^2_{GP} \to G^2_{GP}$.

$$\begin{aligned} \texttt{rec}_{GP}(t,u) &:= (t[\![i \mapsto u_{[j]}]\!],\ u[\![j \mapsto t_{[i]}]\!]) \\ i &:= r_{du}(\{0,|t|\}) \\ j &:= r_{du}(\{0,|t|\}) \end{aligned} \tag{2.14}$$

In this definition, $t_{[i]}$ denotes the ith subexpression of a symbolic expression t, where $t_{[0]} := t$. The notation $t[\![i \mapsto u]\!]$ denotes the expression t whose ith subexpression has been replaced by the symbolic expression u. The number of subexpressions of an expression is written as $|t|$. The standard crossover operator randomly selects a crossover point in each parent expression and swaps the corresponding subexpressions, returning two child expressions.

Other Recombination Operators In standard GP crossover, crossover points are drawn from a uniform random distribution. As the majority of subexpressions are input variables or constants, most standard crossover events affect "deep" subexpressions. This imbalance has lead to the development of several alternative GP recombination operators, including uniform crossover, context-preserving crossover, and size-fair crossover. Poli and Langdon [1998] and Poli et al. [2008] provide a survey of these alternatives.[59]

[59] Poli, R. and Langdon, W. B. (1998). On the ability to search the space of programs of standard, one-point and uniform crossover in genetic programming. Technical Report CSRP-98-7, School of Computer Science, University of Birmingham, U.K; and Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza)

2.6.4 *Selection Operators*

Selection operators are used to select m GP solutions, or individuals, from a pool of n individuals for recombination and mutation (parent selection) or for survival and transfer into the next generation (survivor selection) based on their fitness values. The task of selection is therefore to steer the evolutionary search process towards solutions of high fitness without quickly loosing solution diversity. Thus, the decision of which individuals to select for variation or survival incurs a trade-off between *exploiting* existing good solutions or *exploring* new regions of the search space by also selecting solutions of mediocre fitness.

Selection operators can be deterministic or probabilistic, and single- or multi-objective. Furthermore, selection operators can be put on a gradient between *non-elitist* and *elitist* operators, depending on how much they emphasize exploitation over exploration. In traditional GP systems, single-objective (probabilistic) tournament selection seems to be the most commonly employed selection operator.[60]. In modern GP systems that are mainly used for symbolic regression, multi-objective selection seems to be rapidly gaining popularity. In GP, selection operators with at least some degree of elitism are common.

[60] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza)

Single-Objective Selection In principle, all known single-objective selection operators known in the field of EC can be directly applied to GP. De Jong [2006] gives a thorough overview on single-objective selection operators and their respective strengths and weaknesses.[61] Historically, single-objective tournament selection has been the most commonly used selection operator in GP. Figure 2.17 of Subsection 2.9.1 gives a pseudo-code implementation of this operator. The popularity of tournament selection in the field of GP is rooted in three advantageous properties of this operator: 1) In tournament selection, an individual is selected over another individual if it has higher fitness, regardless of the absolute fitness difference, i.e., tournament selection is a rank-based operator. This leads to automatic rescaling of fitness values and therefore to constant selection pressure. Constant selection pressure is beneficial because it keeps high-fitness individuals from taking over the population in the initial stages of a GP run while also enabling progress in late stages of a run, when absolute fitness value differences are small. 2) By adjusting the tournament size parameter, the degree of elitism of the operator can be controlled. 3) Tournament selection is efficiently implementable.

[61] De Jong, K. A. (2006). *Evolutionary computation - A unified approach*. MIT Press, Cambridge, MA

Multi-Objective Selection As mentioned in Section 2.5.3, there are multiple factors contributing to the overall quality of a GP solution. In addition to solution accuracy, solution compactness has to be taken into account to counter bloat. In data-driven GP, overfitting can also be a problem that can be countered by measuring solution generalizability and taking generalizability into account in the selection step. Thus, a GP fitness value can be regarded as a vector with multiple components. Single-objective selection operators cannot be applied to fitness vectors, which need to be reduced to scalar values, which is most often accomplished by calculating a weighted sum of the components. Unfortunately, it is not trivial to choose weights that give good results. Even more importantly, not all efficient solutions can be found that way.

An attractive alternative is to employ a multi-objective selection operator that can handle fitness vectors directly. In GP, the most common approaches to multi-objective selection are lexicographic selection and Pareto selection. Lexicographic selection uses a lexi-

cographic ordering of the fitness vectors. This implies that fitness components can be ordered according to their importance to overall solution quality. Luke and Panait [2002] introduce a lexicographic GP selection operator that prefers solutions of lower complexity only if they have accuracy values of equal rank.

Pareto selection operators depend on the notion of *Pareto-dominance* as defined in Equation 2.15. A fitness vector x Pareto-dominates a fitness vector x', written as $x \prec x'$, if each component of x is less than or equal to the corresponding component of x' and at least one component is strictly less, assuming that fitness is to be minimized.

$$x \prec x' :\Leftrightarrow \forall i : x_i \leq x'_i \text{ and } \exists j : x_j < x'_j \tag{2.15}$$

A fitness vector x is called *Pareto optimal* if it is a minimal element of the Pareto dominance strict partial order (cf. Equation 2.15). These fitness vectors $x|\forall x' : x \prec x'$ form the first *Pareto front*. Removing all elements of the first Pareto front from the underlying set and recalculating the Pareto front yields the second Pareto front. Repeating this operation leads to a ranking which associates each fitness vector with a Pareto front number. This process is also known as *non-dominated sorting* (NDS).[62] Figure 2.10 shows a Pareto front plot of the result population of an example GP run. Points in the plot denote GP solutions of the first Pareto front. Solution accuracy is shown on the x-axis, solution complexity is shown on the y-axis of the plot.

[62] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA

A Pareto selection operator selects elements from consecutive Pareto fronts until a target number of m elements is selected. During this process, it may be the case that there are more elements left in a Pareto front than elements left to be selected. There are different strategies how to determine which Pareto front elements to select in this case, the most common being selection based on crowding distance, as implemented in the NSGA-II algorithm[63], and selection by hypervolume contribution, as implemented by the SMS-EMOA algorithm.[64]

[63] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197

[64] Beume, N., Naujoks, B., and Emmerich, M. (2007). SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669

Pareto selection operators have a number of advantages over single-objective selection as well as over other multi-objective selection operators. Users are presented with a selection of solutions to choose from, cf. Figure 2.10. Here, users can choose their own trade-off between solution accuracy and solution complexity. They may even choose to use multiple solutions in an ensemble. Basing selection decisions on multiple objectives also helps to preserve solution diversity during a GP run. For these reasons, Pareto selection operators are gaining popularity in modern GP systems.[65] All real-world case-studies discussed in this work employ Pareto selection.

[65] Smits, G. and Kotanchek, M. (2004). Pareto-front exploitation in symbolic regression. In O'Reilly, U. et al., editors, *Genetic Programming Theory and Practice II*, Genetic and Evolutionary Computation Series, pages 283–299. Springer, New York; and Schmidt, M. D. and Lipson, H. (2010). Age-fitness Pareto optimization. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation (GECCO 2010)*, pages 543–544, New York. ACM Press

Population Diversity Preservation Loss of solution diversity in a population, on a genotypic or phenotypic level, leads to premature convergence of a GP run to local optima. GP algorithms can be im-

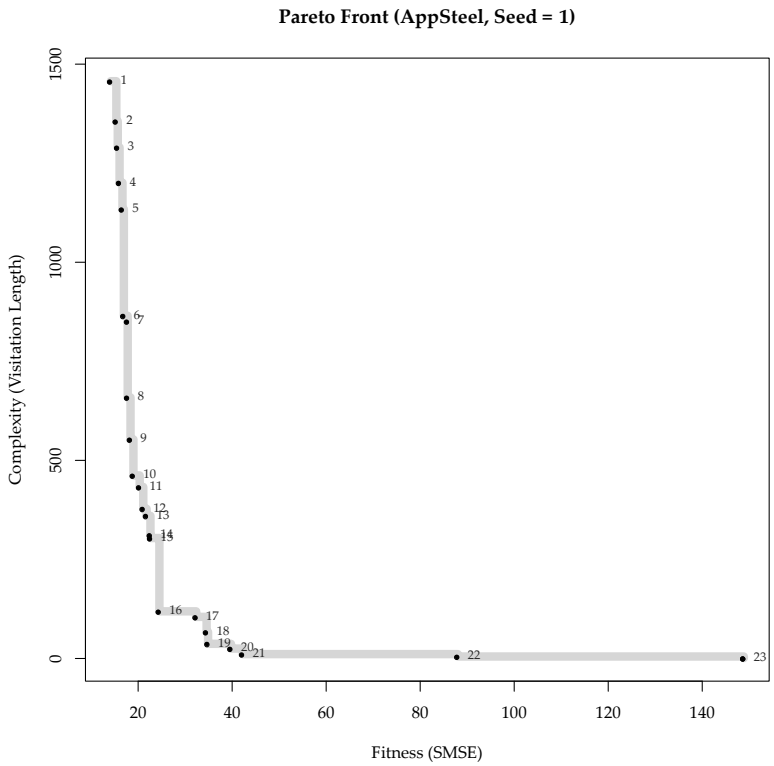


Figure 2.10: Pareto front plot of the result population of an example GP run. Points in the plot denote GP solutions of the first Pareto front. Solution accuracy is shown on the x-axis, solution complexity is shown on the y-axis.

proved by implementing measures of population diversity preservation. Multiple techniques have been proposed in the EA community, including run restarts and multiple parallel runs. A particular successful technique is the Age Layered Population Structure algorithm.[66]. This algorithm tracks how long the genetic material of a solution has been present in the population and uses this information to segregate solutions into age-layers that do not compete with each other. Newly initialized solutions are then added in each generation, leading to an algorithm that never completely converges.

When using a multi-objective selection operator, a dynamic variant of the age-layering technique can be realized by using solution age as an additional optimization criterion. Schmidt and Lipson [2010] describe this approach in detail and demonstrate a significant performance advantage when this approach is applied to symbolic regression.Schmidt and Lipson [2010]

Niching is another method for population diversity preservation, which divides the population into independent sub-populations with limited exchange of genetic information. Shir [2012] provides an overview on this method.[67]

[66] Hornby, G. S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In Keijzer, M. et al., editors, *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO 2006)*, volume 1, pages 815–822, Seattle, WA. ACM Press

[67] Shir, O. M. (2012). Niching in evolutionary algorithms. In Rozenberg, G. et al., editors, *Handbook of Natural Computing*, pages 1035–1069. Springer, Berlin

2.7 *Genotypic and Phenotypic GP Search Spaces*

Based on the definitions of abstract genotypic and phenotypic search spaces introduced in Section 2.4.1, this section looks at concrete GP search spaces. Figure 2.11 illustrates the tiered nature of GP search spaces. Based on the search space definition given as an input variable set, a function set, and a set of mutation and recombination operators, the genotypic search space is determined. This genotypic search space can then be mapped to the phenotypic search space through the genotype-phenotype mapping. This space is then mapped to the fitness space via the fitness function. Practical GP selection operators as defined in Section 2.6.4 are operating on the fitness space alone. Mutation and recombination operators, on the other hand, operate on the genotypic space exclusively.

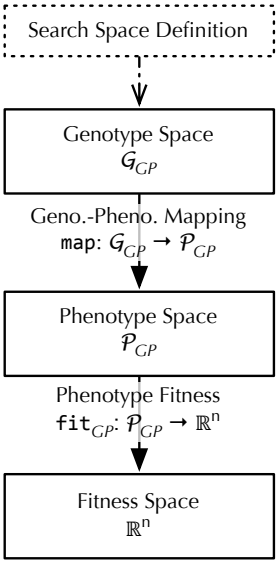


Figure 2.11: The three tiers of GP search spaces. See main text for details.

As an illustration, Figure 2.12 exemplifies instances from the genotypic search space of a typical symbolic regression problem. Solutions are represented as symbolic expression trees. The phenotypic search space with corresponding phenotypic instances is shown in Figure 2.13.

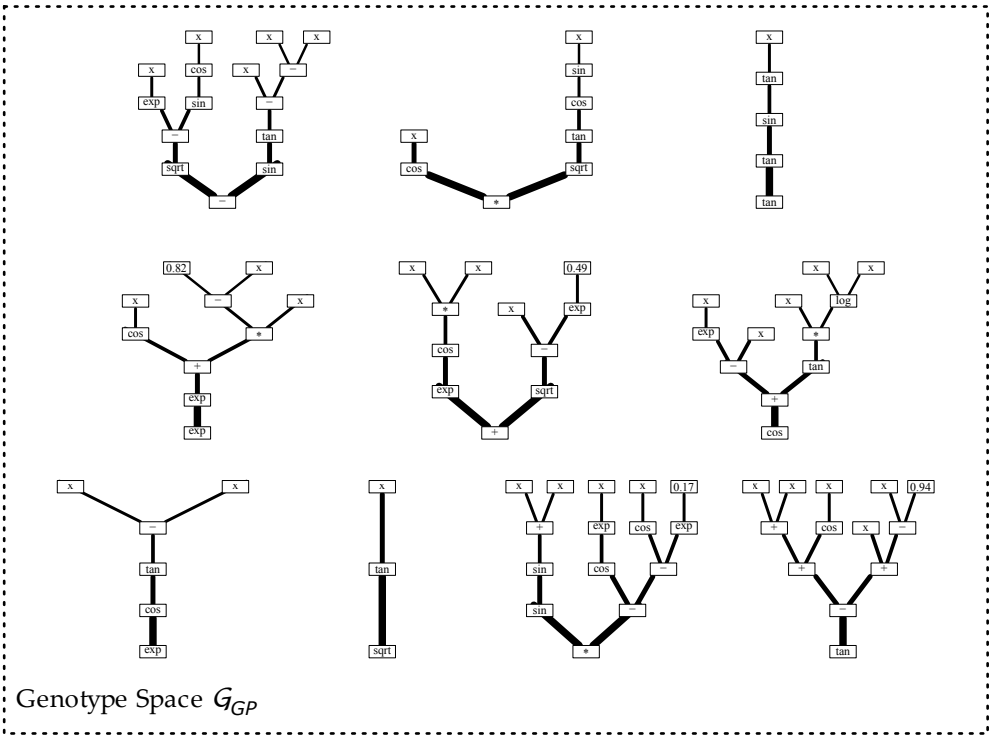


Figure 2.12: Instances of the genotypic search space of a typical symbolic regression problem. Solutions are represented as symbolic expression trees.

The efficiency of evolutionary search is strongly influenced by the ability to find better solutions in the neighborhood of good solutions through mutation, or to find better solutions through recombination of good solutions. Therefore, the genotype-phenotype mapping and the fitness functions should be continuous, meaning that solutions near to each other in genotype space should be near to each other in phenotype space and also be near to each other in fitness space. Ideally, both the genotype-phenotype mapping and

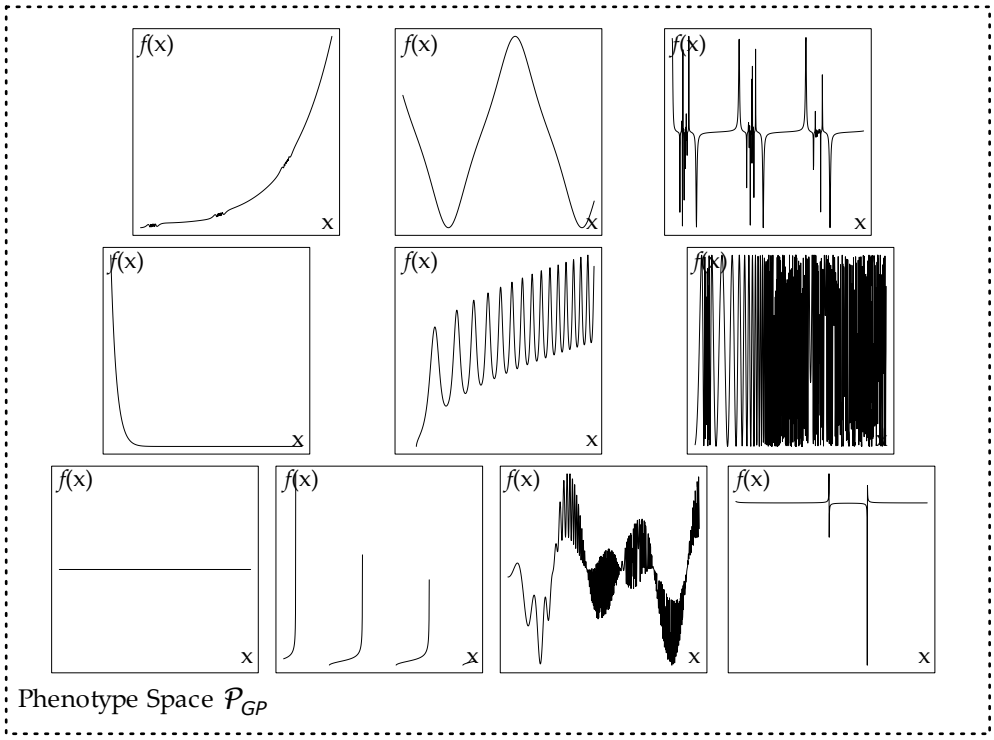


Figure 2.13: Phenotypic search space of a typical symbolic regression problem. The instances shown as members of the space correspond to the instances shown in Figure 2.12.

the fitness function should be a continuous functions. This property is also known as strong causality or search space *locality*. Search space locality can be a useful indicator of GP problem difficulty.

As already mentioned in Section 2.6.2, Droste and Wiesmann [2000] and Moraglio et al. [2012] both introduce design methodologies for GP representations and search operators that induce high locality of the genotype-phenotype mapping and fitness function. Unfortunately, for reasons mentioned above, these methodologies cannot be easily applied to the important problem class of symbolic regression without loosing result compactness and understandability to a large degree. The solution presented by Droste et al. [2000] is elegant and effective, but only applies to symbolic regression of boolean functions. The solution of Moraglio et al. [2012] is effective in theory, but relies on genotypes that grow with each variation step, which renders practical application difficult and leads to large result expressions of low understandability.

Another well-established approach for estimating search space locality is to calculate the *fitness distance correlation*, i.e., the correlation between genotypic distance and absolute fitness value difference.[68] Fitness distance correlation depends on a metric genotype space, a precondition that is seldom fulfilled by representations and operators used in typical modern GP systems. It also depends on a concrete fitness function.

An alternative for locality estimation independent of a concrete fitness function is to estimate the degree of continuity of the genotype-phenotype mapping by correlating genotypic distance,

[68] Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA. Morgan Kaufmann

measured as atomic variation steps, with phenotypic distance, measured by a problem class specific metric, e.g., mean-absolute error for symbolic regression.

In this work, an initial series of experiments has been conducted to demonstrate this method. To estimate the genotype-phenotype distance correlation empirically for the important GP problem class of symbolic regression, the following experiment setup has been used. Based on the GP input variable set x, the function set $\{+, -, *, /, \sin, \cos, \exp, \log, \sqrt{x}\}$, and the constant set $\mathtt{R}$, $s := 100$ symbolic expressions of maximum tree depth $d := 5$ have been sampled via the random initialization operator given in Equation 2.10 (Section 2.6.1). Then, the subtree mutation operator given in Equation 2.12 (Section 2.6.2) with parameters $p := 0.1$, $n := 3$, $p_s := 0.5$, $p_v := 0.5$, $p_i := 0.5$ has been consecutively applied $n \in \{1, \ldots, 5\}$ times to each sampled expression, yielding, for each n, a set of s mutated expressions. Finally, for each set, the phenotypic distance between the original expression and its respective mutant has been measured by calculating the MAE between the phenotypes in the interval $[1, 10]$. In the remainder of this work, this type of experiment will be referred to as *search space locality estimation*.

Figure 2.14 visualizes the results of this experiment. Genotypic distance, measured by number of consecutive mutation operator applications, is shown on the x-axis. Phenotypic distance, measured as MAE between phenotypes, i.e., univariate real-valued functions, in the interval $[1, 10]$, is shown on the logarithmic y-axis. Additionally, the Pearson correlation between genotypic and phenotypic distance is shown in the plot's legend.

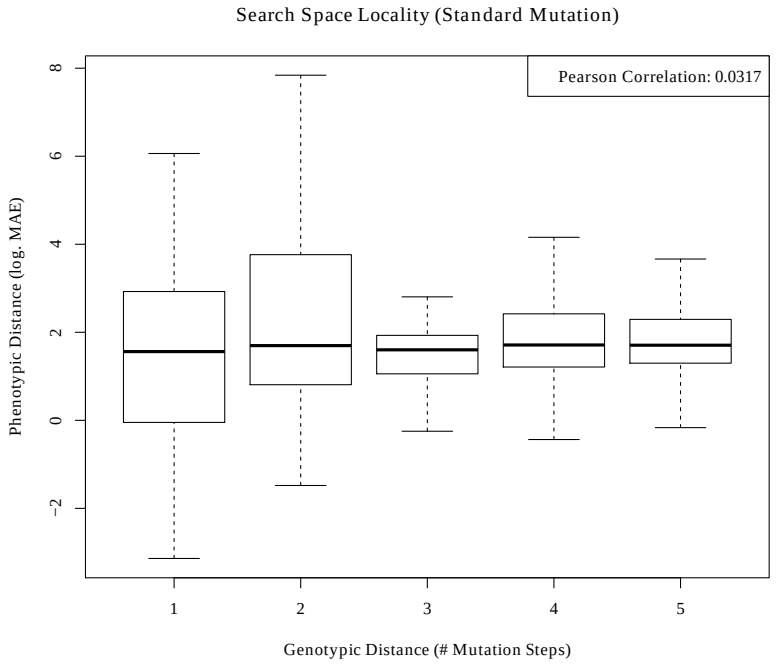


Figure 2.14: Genotype-phenotype distance correlation in standard GP subtree mutation. Genotypic distance, measured by number of consecutive mutation operator applications, is shown on the x-axis. Phenotypic distance, measured as logarithmic MAE, is shown on the y-axis.

Genotypic and phenotypic distances are only very weakly positively correlated with a Pearson correlation value of 0.0317, indicating low locality of the genotype-phenotype mapping. This renders evolutionary search difficult, but not impossible, as results of later experiments will demonstrate (see Chapter 6).

To demonstrate the potential of high-locality search operators, the same experiment for search space locality estimation has been repeated with the geometric semantic GP mutation operator introduced by Moraglio et al. [2012]. This operator, $\texttt{mut}_g :: G_{GP} \times \mathbb{R} \times \mathbb{N} \times \mathbb{P}^2 \rightarrow G_{GP}$, is defined as follows:

$$\texttt{mut}_g(t, s, n, p_s, p_v) := t + s[\texttt{init}(n, p_s, p_v) - \texttt{init}(n, p_s, p_v)]$$

The operator requires $\{+, -, *\}$ to be a subset of the function set, as well as the availability of real-valued constants in the constant set. The operator proceeds by constructing a symbolic expression that subtracts a randomly initialized symbolic expressions of maximum depth n from another randomly initialized symbolic expression of maximum depth n. The result of the subtraction are multiplied by a constant s. This constant s serves as a step size parameter to control mutation strength. In this experiment, it has been set to a value of $s := 0.1$. See Section 2.6.1 for an explanation of the parameters p_s and p_v. Experiment results are shown in Figure 2.15.

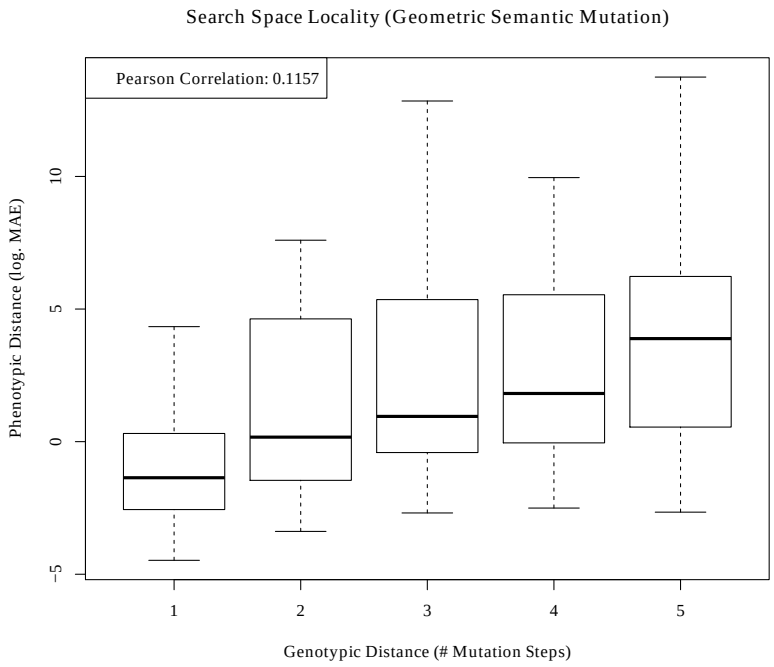


Figure 2.15: Genotype-phenotype distance correlation in geometric semantic GP mutation. The remarks to Figure 2.14 also apply here.

Genotypic and phenotypic distances are positively correlated with a Pearson correlation value of 0.1157, indicating higher locality of the genotype-phenotype mapping compared to the locality induced by a standard GP mutation operator. Unfortunately, using geometric semantic GP mutation and crossover operators comes

with a price. Applying geometric semantic GP variation operators incurs a size increase of the affected symbolic expression. With geometric semantic mutation, size increases is linearly, while geometric semantic crossover incurs exponential growth in symbolic expression size. A GP system using naive implementations of these operators would quickly run out of memory resources before attaining good results for non-trivial problems. At the time of this writing, efficient implementations are in development using techniques such as higher-order functions and memoization of intermediate results, that could make the applications of these operators feasible in practice. However, efficient implementations do not solve the problem of low human understandability of the very complex GP expressions generated by these operators. For this reason, geometric semantic GP operators are not considered further in the remainder of this work.

To summarize, GP representations and operators providing higher locality of the genotype-phenotype mapping can provide significant performance benefits to evolutionary search. Unfortunately, representations inducing high locality sometimes conflict other GP system design goals, such as result understandability. Currently, this seems to be the case with the important GP application class of symbolic regression of real-valued functions.

2.8 *Defining Valid Regions in GP Search Spaces*

There are two main paradigms to defining valid regions in GP search space. The first paradigm completely excludes invalid regions from the search space by construction of the genetic operators. Examples that fit into this category are Strongly Typed GP, Grammatical Evolution, and Repair Functions for GP individuals. In this paradigm, valid regions have sharp margins. In the second paradigm, invalid regions are suppressed by fitness reduction, leading to soft margins. Parsimony pressure is a common technique that follows this paradigm.

Both approaches have distinct benefits and drawbacks and are often combined in practice. Excluding invalid regions leads to smaller search spaces. If there is prior knowledge of the general structure of a valid solution, this reduction of search space size might be significant. Yet, the assumption that smaller search spaces are always more tractable does not always hold. This is because a smaller search space might exclude important middle ground for evolution, i.e., its fitness landscape might be more rough that of a larger superset of this space. There is some experimental evidence that this is often the case.[69]

[69] Keijzer, M. and Babovic, V. (2002). Declarative and preferential bias in GP-based scientific discovery. *Genetic Programming and Evolvable Machines*, 3(1):41–79

2.9 *GP Search Heuristics*

In this work, the term *GP search heuristic* describes the concrete search strategy used in a GP system, which is in principle indepen-

dent of the concrete GP search space. Typically, GP uses a steady state GA with tournament selection to search genotype space. In the GP literature, many different concrete variants have been described. As mentioned, it is possible to de-couple the search heuristic from the search space, giving rise to a wide variety of possible hybrid algorithms. An example is the evolution of support vector machine kernels, which includes a memetic approach of kernel constant optimization. Of course it is entirely possible to use search heuristics from outside the field of EAs in GP search.

Historically, every GP system implemented slightly different search heuristics, while exhaustive comparisons of GP search heuristics, isolated from the concrete GP search space, are scarce. Modern GP systems often employ multi-objective EAs (EMOAs) as search heuristic. For historical reasons and to simplify parallelization on shared memory multiprocessors, steady state algorithms with Pareto tournament selection are the predominant EMOA variants used in today's best-performing GP systems. In simple GP systems mainly designed for research and teaching, single-objective steady state EAs with tournament selection seem to be still very widespread.[70]

[70] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza)

This section describes the set of GP search heuristics examined in this work. For each heuristic, the general algorithmic framework implementing selection, i.e., the concrete implementation of the selection operator `sel` of Equation 2.2, is described. GP search heuristics may be classified as generational, steady state, or generation gap algorithms. The *variation pipeline*, i.e., the concrete setup and order of application of variation operators (recombination and mutation) is given, as some GP search heuristics exclusively use recombination or mutation.[71] Furthermore, the main features of the selection strategy are described, including number and details of selection criteria, as well as the trade-off made between exploration of new search space areas and exploitation of existing solutions. Most modern GP search heuristics make provisions for preserving genetic diversity and related to that, avoiding premature convergence of a population to local optima. These provisions may include fitness sharing, crowding, niching, age layering, or restarts. Diversity preservation approaches implemented are also part of the description of each GP search heuristic. Finally, for each GP search heuristic, parameters including types, ranges, default values, and constraints are presented.

[71] For example, Koza's original GP system lacked mutation operators and relied exclusively on recombination.

The GP search heuristics selected for study in this work can be split into two sets, based on their origin. The first set contains GP search heuristics implemented in existing GP systems, including TinyGP and Eureqa. It is striking that all of these search heuristics are single- or multi-objective steady state EAs with tournament selection. The second set contains new GP search heuristics that use more traditional generational EAs. This set was included to answer the question whether these more traditional and arguably theo-

retically better understood and easier implemented EAs can reach comparable performance when applied to GP. As mentioned above, steady state EAs based on tournament selection are traditionally used in GP systems to simplify parallelization on shared memory multiprocessors. Nonetheless, it is also possible to parallelize generational EAs, while the parallelization on large scale distributed memory multiprocessors is of comparable difficulty with both algorithmic schemes. An overview of the most important features and attributes of the GP search heuristics studied in this work is given in Table 2.1.

	TinyGP	GSOGP	GMOGP
Optimization Criteria	Fitness	Fitness	Fitness, Complexity, Age
Selection Framework	Steady State	Generational ($\mu + \lambda$)	Generational ($\mu + \lambda$)
Parent Selection	Uniform Random	Uniform Random or Rank-based	Uniform Random or NDS
Variation Pipeline	`rec` $\rightarrow$ `mut`	`rec` $\rightarrow$ `mut`	`rec` $\rightarrow$ `mut`
Survivor Selection	Rank-based	Rank-based	NDS
Diversity Preservation	-	-	Age-Fitness Pareto Optimization (AFPO)

Table 2.1: Overview of the most important features and attributes of the GP search heuristics described in this work.

2.9.1 *TinyGP*

TinyGP is a popular small GP implementation mainly used in teaching. It implements a simple steady-state single-objective search heuristic with tournament selection that is loosely based on Koza's original work on GP.[72] Steady-state search heuristics with tournament selection are very popular in GP, both for simple teaching systems as well as for complex real-world systems, as they are relatively simple to implement and allow straight-forward parallelization. The TinyGP search heuristic can be seen as a deliberately minimal single-objective example for the popular class of steady-state GP search heuristics with tournament selection. For this reason, it was implemented in RGP and included in this study as a baseline.

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

Algorithm Structure TinyGP employs a simple single-objective steady state EA as its search search heuristic. In the first step of the algorithm, a population $\texttt{pop}(0)$ of μ random individuals is created. Next, the steady-state evolution process starts by randomly selecting either a recombination or a mutation operator. The probability for selecting the recombination operator is given by the parameter $p_{\texttt{rec}}$. In case of recombination, the algorithm selects two parents via two independent tournaments of size $s_{\text{tournament}}$ as detailed in the next paragraph. Note the non-zero probability of choosing the same individual for both recombination parents, as tournaments are performed independently. In case of mutation, a single parent is chosen in a single tournament. In both cases, a single child is creating by applying the chosen variation operator to the parent(s). Next, the algorithm chooses a individual to replace by this child in

a single negative tournament of size $s_{\text{tournament}}$. This process is repeated until a predefined termination criterion is met. Pseudo-code for this search heuristic is shown in Figure 2.16.

```
pop ← createIndividuals(number = μ)

while (termination criterion not met) {
  child ← if (randomUniformNumber() ≤ p_rec) {
    mother ← tournament(pop, s_tournament)
    father ← tournament(pop, s_tournament)
    rec(mother, father)
  }
  else {
    parent ← tournament(pop, s_tournament)
    mut(parent)
  }

  replaced ← negativeTournament(pop, s_tournament)

  pop[replaced] ← child
}

return(pop)
```

Figure 2.16: Pseudo-code implementation of the TinyGP search heuristic. ω_{RNG} is the probability space used to model the random number generator function `randomUniformNumber`, which generates uniform distributed random numbers in the closed interval $[0, 1]$. The recombination operator `rec` must be defined on pairs of individuals.

Selection Strategy Tournament selection in TinyGP proceeds as follows: First, an individual is selected from the population by uniform random sampling as the current best individual. This individual's fitness is then compared to competitors $s_{\text{tournament}}$ times. Each time a competitor has a better (smaller) fitness value, it takes the place as the current best individual. Competitors are chosen by uniform random sampling from the entire population. Therefore, there is a non-zero probability that the same individual enters the same tournament multiple times.

The negative tournament selection operator employs the same strategy, its only difference being that the order relation $<$ is being replaced by its converse $>$, so that the worst individual taking part in the tournament is returned as result.

Real-world implementations of the tournament selection operator often include extensions and optimizations. For example, individuals participating in a tournament are often sampled beforehand, without replacement, avoiding the inefficiency of duplicated individuals in tournaments.

Figure 2.17 gives a pseudo-code implementations of the tournament and negative tournament selection operators described above and referred to in Figure 2.16.

Diversity Preservation The TinyGP system does not implement any internal means of diversity preservation, but can be extended with well-known external measures, such as fitness sharing, crowding, niching, and automatic restarts without much effort. For simplicity,

```
tournament ← function(pop, s_tournament) {
  bestIndividual ← sampleWithoutReplacement(pop,
                     number = 1)
  bestFitness ← ∞

  for (i in 1:s_tournament) {
    competitor ← sampleWithoutReplacement(pop,
                   number = 1)
    if (fit(competitor) < bestFitness) {
      bestFitness ← fit(competitor)
      bestIndividual ← competitor
    }
  }

  return(bestIndividual)
}

negativeTournament ← function(pop, s_tournament) {
  worstIndividual ← sampleWithoutReplacement(pop,
                      number = 1)
  worstFitness ← ∞

  for (i in 1:s_tournament) {
    competitor ← sampleWithoutReplacement(pop,
                   number = 1)
    if (fit(competitor) > worstFitness) {
      worstFitness ← fit(competitor)
      worstIndividual ← competitor
    }
  }

  return(worstIndividual)
}
```

Figure 2.17: Pseudo-code implementation of tournament selection.

these extensions were not implemented and not included in this study.

Parameters Table 2.2 lists all parameters of the TinyGP search heuristic. Note the comparatively large default population size, which is typical for classical steady state GP search heuristics.

	Variable (Symbol)	*Domain*	*Default*
Population Size	`mu` (μ)	$\mathbb{N}$	300
Tournament Size	`tournamentSize` ($s_{\text{tournament}}$)	$\mathbb{N}$	2
Recombination Probability	`recombinationProbability` ($p_{\texttt{rec}}$)	$[0, 1]$	0.9

Table 2.2: Parameters of the TinyGP search heuristic.

There are no additional hard parameter constraints in the RGP implementation of the TinyGP search heuristic, although the tournament size $s_{\text{tournament}}$ should be smaller than or equal to the population size μ.

2.9.2 *Generational Single-Objective GP*

Generational Single-Objective GP (GSOGP) is a very simple generational single-objective GP search heuristic modeled after a classical single-objective ES with rank-based selection, as described by De Jong [2006].[73] Because single-objective generational search heuristics were never commonly used in GP for scalability reasons, and single-objective search heuristics are at least partially superseded by multi-objective variants in state-of-the-art GP systems, this heuristic is included mainly as a baseline. Every search heuristic described in this study, with exception to the TinyGP search heuristic which allows straight-forward parallelization, should be at least as effective.

[73] De Jong, K. A. (2006). *Evolutionary computation - A unified approach*. MIT Press, Cambridge, MA

Algorithm Structure As indicated above, GSOGP constitutes a classical single-objective generational $(\mu + \lambda)$ evolution strategy. During initialization, a population $\texttt{pop}(0)$ of μ random individuals is created. The iterative evolution process starts by selecting λ pairs of parents via uniform random sampling without replacement. A recombination operator is then applied to each parent pair, yielding λ children. A mutation operator is then applied to each child. Next, μ individuals are chosen from the $(\mu + \lambda)$-sized set union of parents and mutated children by rank-based selection. These individuals replace the parent population. The iterative process is stopped when a predefined termination criterion is met. Figure 2.18 gives pseudo-code of the GSOGP search heuristic. Note that a real implementation would retain final fitness values together with the final population, eliminating the need to re-evaluate the fitness of each individual result presentation.

Selection Strategy The selection operator $\texttt{sel}\langle \mathit{GSOGP} \rangle$ implements the well-known rank-based selection scheme. Individuals are

```
pop ← createIndividuals(number = μ)

while (termination criterion not met) {
  parents ← sampleWithoutReplacement(pop,
                number = 2 × λ)
  mothers ← parents[1:λ]
  fathers ← parents[(λ+1):2×λ]
  children ← mut_pop(rec_pop(mothers, fathers))

  selectionPool ← parents ∪ children
  survivors ← sel⟨GSOGP⟩(selectionPool, number = μ)
  pop ← survivors
}

return(pop)
```

Figure 2.18: Pseudo-code implementation of the GSOGP search heuristic. The recombination operator `rec` must be defined on pairs of individuals.

ranked by their fitness function values, then then n best individuals are selected according to this ranking. Some extended variants of rank-based selection assign selection probabilities according to the fitness-based ranking and use weighted sampling to give individuals below rank n the chance of being selected for survival. For reasons of simplicity and interpretability of results, these extensions were not implemented in RGP and not considered in this work.

Diversity Preservation As the TinyGP search heuristic, the GSOGP search heuristic does not implement any internal means of diversity preservation, but can just as easily be extended with well-known external measures, such as fitness sharing, crowding, niching, and automatic restarts. For simplicity, these extensions were not included in this work, although a cluster-based niching approach in implemented in the RGP system.

Parameters All parameters of the GSOGP search heuristic are listed in Table 2.3.

	Variable (Symbol)	*Domain*	*Default*
Population Size	`mu` (μ)	$\mathbb{N}$	100
Children per Generation	`lambda` (λ)	$\mathbb{N}$	50
Parent Selection Probability	`parentSelectionP` (p_{psel})	$[0,1]$	1

Table 2.3: Parameters of the GSOGP search heuristic.

In the RGP implementation of this search heuristic, these parameters are subject to the following constraint:

$$\lambda \leq \left\lfloor \frac{\mu}{2} \right\rfloor \qquad \text{(Children Set Size)}$$

As parents are sampled without replacement from an uniform distribution and two parents are needed for recombination, the number of children per generation must be smaller than or equal to half the population size.

2.9.3 Generational Multi-Objective GP

Generational Multi-Objective GP (GMOGP) is a generational multi-objective GP search heuristic that combines ideas of state-of-the-art multi-objective GP search heuristics with design concepts of modern generational multi-objective EAs. The main reason for its inception and inclusion in the set of search heuristics studied is to answer the research question of whether it is possible to reach performance comparable to Ordinal Pareto GP (OPGP) with an conceptually simpler generational multi-objective GP search heuristic.[74]

[74] Smits, G. and Vladislavleva, E. (2006). Ordinal Pareto genetic programming. In Yen, G. G. et al., editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 3114–3120, Piscataway, NJ. IEEE Press

Algorithm Structure In contrast to OPGP and as its name implies, GMOGP is based on a classical generational $(\mu + \lambda)$ strategy. After creating an initial population $\texttt{pop}(0)$ of μ random individuals, the iterative evolution process starts by choosing λ pairs of parents either with probability p_{psel} by the Pareto selection operator detailed in the next paragraph, or with probability $1 - p_{\text{psel}}$ by uniform random sampling without replacement. Pairwise recombination is applied before mutation, yielding λ children. Next, μ individuals are chosen from the $(\mu + \lambda + \nu)$-sized set union of parents, children and ν newly initialized individuals by the Pareto selection operator detailed in the next paragraph, replacing the parent population. This iterative process is stopped when a predefined termination criterion is met. Figure 2.19 outlines the GMOGP search heuristic in pseudo-code.

```
pop ← createIndividuals(number = μ)

while (termination criterion not met) {
  parents ← if (randomUniformNumber() ≤ p_psel) {
    sel⟨GMOGP⟩(pop, number = 2 × λ)
  }
  else {
    sampleWithoutReplacement(pop, number = 2 × λ)
  }
  mothers ← parents[1:λ]
  fathers ← parents[(λ+1):2 × λ]
  children ← mut_pop(rec_pop(mothers, fathers))
  newIndividuals ← createIndividuals(number = ν)

  selectionPool ← parents ∪ children ∪ newIndividuals
  survivors ← sel⟨GMOGP⟩(selectionPool, number = μ)
  pop ← survivors
}

return(pop)
```

Figure 2.19: Pseudo-code implementation of the GMOGP search heuristic. The recombination operator `rec` must be defined on pairs of individuals.

Selection Strategy The selection operator $\texttt{sel}\langle GMOGP\rangle$ is based on NDS of the selection pool based on the three criteria fitness, geno-

typic complexity, and *genotypic age* (see next paragraph). Ties in the NDS are broken by crowding distance (CD). Thus, the selection strategy matches the selection strategy of the well-established NSGA-II EMOA.

Diversity Preservation GMOGP implements elements of Schmidt and Lipson [2010]'s Age-Fitness Pareto Optimization (AFPO) algorithm for preserving genotypic diversity and avoiding premature convergence. [75] In each generation, a fixed number of newly initialized individuals are inserted into the population to maintain genetic diversity. Of course, these new individuals will be of low fitness on average and would be quickly selected for replacement without having the chance to obtain local optima through a series of variation steps. This problem is countered by the introduction of genotypic $\texttt{age} : \mathcal{G} \to \mathbb{N}$, as defined as follows:

[75] Schmidt, M. D. and Lipson, H. (2010). Age-fitness Pareto optimization. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation (GECCO 2010)*, pages 543–544, New York. ACM Press

$$\begin{aligned} \texttt{age}(g_{new}) &:= 0, \\ \texttt{age}[\texttt{mut}(g)] &:= \texttt{age}(g) + 1, \\ \texttt{age}[\texttt{rec}(g_A, g_B)] &:= \max[\texttt{age}(g_A), \texttt{age}(g_B)], \end{aligned} \tag{2.16}$$

where g_{new} is a new genotype just inserted into the selection pool, and g, g_A, and g_B are individuals already existing in a population. The genotypic age of a new individual is defined as 0, mutation of an existing individual increases its age by one, and the age of the recombination of two parents is the maximum of their ages. Therefore, the age of an individual is the age of its oldest ancestor, even if no genetic material of that ancestor is left. Genotypic age is then considered as another optimization criterion to minimize, in addition to the minimization criteria fitness and genotypic complexity. This leads to an emergent dynamic age-layering of the population, as young individuals are not dominated by older more fit or less complex genotypes. Therefore, younger individuals are allowed to evolve independently of older individuals, until they reach the same age, i.e., have been subject to the same number of variation steps. Ideally, this approach should preserve genetic diversity and enable the discovery of new local and global optima throughout the entire duration of a GP run. The diversity preservation mechanism can be disabled by setting the boolean search heuristic parameter *Age Layering* to `false`.

Parameters Table 2.4 presents all parameters of the GMOGP search heuristic.

	Variable (Symbol)	*Domain*	*Default*
Population Size	`mu` (μ)	$\mathbb{N}$	100
Children per Generation	`lambda` (λ)	$\mathbb{N}$	50
New Individuals per Generation	`nu` (ν)	$\mathbb{N}_0$	50
Age Layering	`ageLayering`	$\mathbb{B}$	`true`
Parent Selection Probability	`parentSelectionP` (p_{psel})	$[0, 1]$	1

Table 2.4: Parameters of the GMOGP search heuristic.

As in the GSOGP search heuristic and for the same reason, these parameters are subject to the following constraint:

$$\lambda \leq \left\lfloor \frac{\mu}{2} \right\rfloor \qquad \text{(Children Set Size)}$$

2.10 Conclusions

The main theme of this chapter was the introduction of a formal framework for GP that is able to succinctly describe a wide array of different GP algorithms. This framework was then put to use for defining important components of classical and modern GP systems in an implementation-independent manner, leading to some observations about the impact of the frameworks design to its ease of use.

Separation of Concerns A recurring theme in the framework is the separation of concerns, as exemplified by the separation of problem and search space definition from search heuristics. This enables all search heuristics defined in the framework to work on all problems and search spaces defined in the framework without manual adaption, simplifying component definition, component implementation, and experimentation.

Reusability and Modularity The principle of separation of concerns described in the last section also aids reusability and modularity. For example, new individual representations can be quickly prototyped by reusing all other GP components, including variation and selection operators, because all components are defined in terms of abstract concepts that easily map to clearly defined software interfaces.

Modern GP *Search Heuristics* Modern GP search heuristics, such as the GMOGP heuristic defined in Section 2.9.3 are promising to solve longstanding problems of GP, including the problem of bloat and premature convergence due to population diversity loss. The empirical study presented in Chapter 5 will put this promise to a series of tests.

3
A Modular GP Implementation Based on R

This chapter describes RGP, a modular GP system that provides the basis for most experimental work described in this thesis. RGP is based on the R environment for statistical computing. The system implements classical untyped tree-based GP as well as more advanced variants and search heuristics including, for example, strongly typed GP and Pareto GP. It strives for high modularity through a consistent architecture that allows the customization and replacement of every algorithm component, while maintaining acceptable performance and accessibility for new users by adhering to the *convention over configuration* principle.

Typical GP applications are supported by standard R software interfaces. For example, symbolic regression via GP is supported by the same *formula interface* as linear regression in R. RGP is freely available as an open source R package from CRAN.[1]

[1] see http://cran.r-project.org/web/packages/rgp/

After a comparison with other GP systems, this chapter provides an overview of RGP's feature set and describes common use cases by means of examples. Although this chapter is not meant as a reference manual, RGP's feature set is described comprehensively. A reference manual detailing every feature is available at http://cran.r-project.org/web/packages/rgp/rgp.pdf or through R's online help system. This chapter also serves as a guide and tutorial for configuring and adapting RGP to concrete use cases. Peripheral tasks not directly tied to GP but important to the modeling process, such as data import, data preprocessing, and result analysis, are also described in appropriate detail. Furthermore, RGP's design is motivated both from a user's perspective as well as from a software engineering perspective.

3.1 Other GP Systems

Fueled by the rapid increase of data volume collected in nearly every field of science and engineering and the need to exploit these data for forecasting and optimization, coupled with the commodification of cheap processing power through cloud service providers, interest in data-driven GP, and in symbolic regression in particular, has grown rapidly. A new generation of symbolic regression

software has been introduced, focusing on user-friendliness and applicability to real-world tasks. Commercial vendors of these systems are growing quickly, as demonstrated by the examples of Nutonian[2] and Evolved Analytics[3].

[2] see http://www.nutonian.com/products/eureqa/

[3] see http://www.evolved-analytics.com/?q=datamodeler

TABLE 3.1 shows a feature-comparison matrix of the three most prevalent GP systems and RGP. In this table, the *General* section should be self-explanatory except for the *Cost per Seat* row: Some vendors offer feature-reduced versions of their systems for a lower price, these have been omitted for brevity.

The *Application Area Support* section lists application classes that are directly supported through specialized tools and user interfaces. *Unsupervised Learning* means the possibility to induce relations from input data without defining dependent variables, as introduced by Schmidt and Lipson [2009].[4]. *Custom* denotes the possibility to provide a custom fitness function in source code form.

[4] Schmidt, M. D. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85

In the *User Interface* section, *Scripting* means the possibility to fully automate symbolic regression runs through an interactive scripting language. *Embedding* means the ability for a system to be embedded in other software as a library.

In the *Data Preprocessing* section, *ETL* means the availability of non-trivial tools for data extraction, transformation, and loading. *EDA* denotes the availability of non-trivial tools for exploratory data analysis, such as multiple data-visualizations and descriptive statistics.[5]

[5] Tukey, J. W. (1977). *Exploratory Data Analysis*. Behavioral Science: Quantitative Methods. Addison-Wesley, Reading, MA

The *Algorithm* section contains features pertaining to the GP algorithm itself. *Optimization* shows whether the GP algorithm supports multi-objective selection or is single-objective only. Support for complexity, diversity, and linearity optimization criteria is listed next. *Search Strategy* shows whether the algorithm's search strategy is modular, i.e., replaceable without affecting the remainder of the system, or fixed. *Type System* denotes support for strongly typed GP. *Ensemble Support* means the possibility to use ensembles of symbolic expressions as models, in contrast to single symbolic expressions only. *Parameter Tuning* means the ability for automatic tuning of algorithm parameters. *HPC Support* denotes the support for GP runs on high-performance parallel computing resources.

The *Result Analysis* section lists features for presenting, analyzing, and deploying GP run results. A *Model Table* presents all models resulting from a GP run, together with relevant meta-data such as validation fitness and complexity values, in tabular form. *Pareto Plots* present the Pareto front of the set of result models of a GP run, cf. Figure 2.10. *Variable Presence Plots* show the distribution of input variables used in the set of result models of a GP run. *Model Simplification* describes the ability to simplify model expressions without changing their semantics through computer algebra methods. *Report Generation* means the ability to automatically generate documents summarizing the results of a GP run.

	RGP	ECJ	DataModeler	Eureqa
General				
Release	0.4-0	21	8.16	0.99.9
Author	Oliver Flasch et al.	Sean Luke et al.	Evolved Analytics LLC	Nutonian, Inc.
Platform	R	proprietary (Java)	Mathematica	proprietary (C++)
License	open source	open source	commercial	commercial
Cost per Seat	–	–	$5,000/year	$2,499/year
Commercial Support	✓	–	✓	✓
Application Area Support				
Regression	✓	✓	✓	✓
Classification	✓	✓	✓	✓
Unsupervised Learning	–	–	–	✓
Feature Selection	✓	–	✓	–
Active DoE	–	–	✓	–
Business Intelligence	–	–	–	✓
Custom	✓	✓	✓	–
User Interface				
Graphical	✓ (via RGP UI)	– (deprecated)	–	✓
Command Line	✓ (R)	✓	✓ (Mathematica)	–
Scripting	✓ (R)	–	✓ (Mathematica)	–
Embedding	✓ (R)	✓	✓ (Mathematica)	✓ (C++ API)
Data Preprocessing				
ETL	✓	–	✓	–
EDA	✓	–	✓	✓
Outlier Handling	✓	–	✓	✓
Algorithm				
Optimization	multi-objective	single-objective	multi-objective	multi-objective
Complexity Crit.	✓	–	✓	✓
Diversity Crit.	✓	–	✓	✓
Linearity Crit.	–	–	✓	–
Search Strategy	modular	fixed	modular	fixed
Type System	✓	✓	✓ (patterns)	–
Ensemble Support	–	–	✓	–
Parameter Tuning	✓ (via SPOT)	–	–	–
HPC Support	✓ (via Rmpi)	–	✓ (Mathematica)	✓ (proprietary)
Result Analysis				
Model Tables	✓	–	✓	✓
Pareto Plots	✓	–	✓	✓
Variable Presence Plots	✓	–	✓	–
Model Simplification	✓ (via Rrules)	–	✓	✓
Report Generation	✓ (via knitr)	–	✓	✓

Table 3.1: Feature comparison matrix for modern GP systems. The information presented is based on publicly available documentation as of January 2014. See the main text for explanations regarding the features listed in this table.

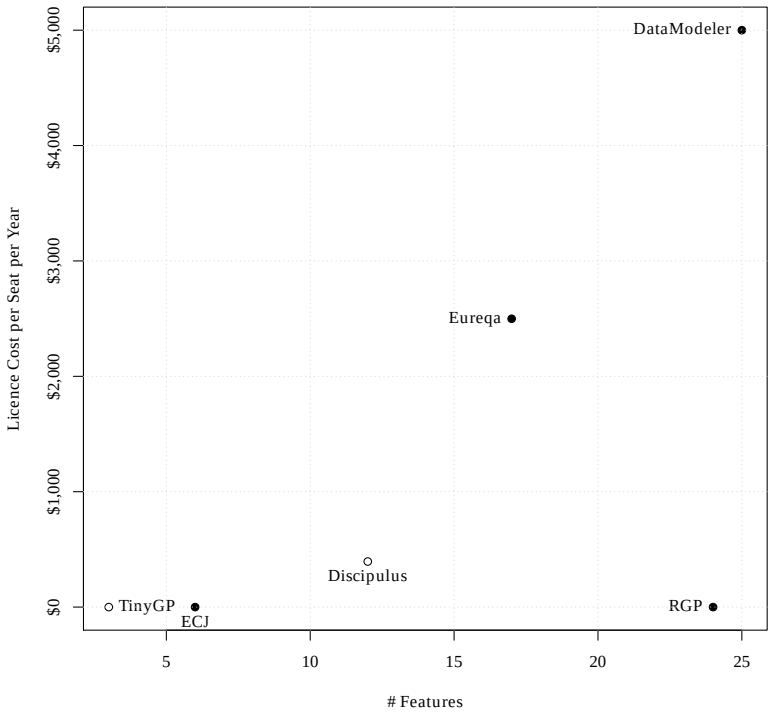


Figure 3.1: Features versus costs of modern GP system offerings. Number of features as of Table 3.1 are shown on the x-axis, license cost per seat and year are shown on the y-axis. In addition to the systems included in Table 3.1 and plotted as solid dots, the lesser known Discipulus system and the well known but but very basic TinyGP system are also shown, plotted as hollow dots. Note that this plot is naturally biased towards the features listed in Table 3.1. Also, the dimension of algorithm efficiency is not included in this comparison due to lack of a thorough experimental study. Nonetheless, both Eureqa and Discipulus are known to have highly optimized implementations.

ECJ ECJ is an open source EC framework written in the Java programming language.[6] According to its authors, the system was designed for large, heavy-weight experimental needs. Included in the framework are implementations of many popular EC algorithms, with an emphasis towards GP. ECJ was developed as a successor to the early GP system lil-gp.[7] Additional features include evolutionary multi-objective optimization algorithms, co-evolution, island models for diversity preservation and parallel computation, parsimony pressure techniques for solution complexity control, multiple individual representations including trees and rule-sets, and multiple search heuristics, including steady-state EA, GA, and ES. While multi-objective selection operators are included in the framework, the GP code uses single-objective selection only. Source code for a basic graphical user interface is included, but its use in the current version is discouraged by the authors. The framework is based on a complex object-oriented design that includes proprietary base classes for all important concepts in EA, such as `Population`, `Individual`, `Evaluator`, `Fitness`, or `Problem`. The system is designed to be modular, extensible, and highly configurable, with all system settings exposed in a large parameter file. As the authors of ECJ assert, the system has a steep learning curve and the initial development overhead for starting a new project is relatively large. This might well be attributed to shortcomings of the object-oriented paradigm as implemented by the Java language in general, instead

[6] Luke, S. (2013). The ECJ owner's manual – A user manual for the ECJ evolutionary computation library. 21th edition, http://cs.gmu.edu/ eclab/projects/ecj/docs/manual/manual.pdf (retrieved 20.02.2015)

[7] Zongker, D. and Punch, B. (1996). *lil-gp 1.01 User's Manual*. East Lansing, MI. Michigan State University

of to shortcomings of ECJ's design in particular. Notwithstanding, thanks to its broad feature set, stable and reasonably efficient implementation, and long development history, the system is widely used in the GP community and also popular in the larger EA community. Based on number of citations, it might be the most popular GP system today.

DataModeler The commercial GP system DataModeler[8], developed by Evolved Analytics LLC, is based on the commercial computer algebra system Mathematica. Early versions were based on research conducted at an international chemical corporation, explaining the systems focus on real-world applicability. In contrast to ECJ, DataModeler is a purely a GP system, with a strong emphasis on symbolic regression. Being based on Mathematica brings advantages and disadvantages: DataModeler has by far the most advantaged symbolic expression manipulation and simplification tools of all GP systems described in this section. The Mathematica Notebook user interface provides very powerful tools for data preprocessing, result analysis, visualization, and the generation of interactive reports. On the other hand, Mathematica incurs considerable licence costs for commercial users, in addition to the licence costs of DataModeler itself. Also, Mathematica has a steep learning curve itself, especially for users from fields like engineering, where it is less commonly deployed. Of course, the same argument can be made for other platforms, such as R or MATLAB. Although the system is very well documented, its lack of a simple graphical user interface can be a hurdle for novice or occasional users. A unique characteristic of DataModeler is its support for ensemble models. By using an ensemble of structurally different high-quality models, the system gains the ability to associate confidence values to predictions based on prediction agreement of the models in the ensemble. The system has a richer feature set than all other GP systems described in this section and might be the most feature-complete implementation of symbolic regression today. Unique characteristics of DataModeler include the support for *function patterns* as an easy-to-use alternative to conventional strongly typed GP, very detailed interactive result reports, and support for *Active DoE*, a design of experiments variant based on symbolic regression.

[8] see http://www.evolved-analytics.com

Eureqa Eureqa[9] is a commercial GP system developed by Nutonian, Inc. and, like DataModeler, focused primarily on symbolic regression.[10] Unlike the Mathematica-based DataModeler system, Eureqa is based on a proprietary platform written in C++. The system is available in desktop and server editions, where desktop editions have a graphical user interface that covers all functionality while still seeming intuitive to occasional users. Special user interfaces for business users are also available as additional purchases, including an integration layer for the popular spreadsheet software Microsoft Excel, as well as a web-based user interface for large scale

[9] see http://www.nutonian.com

[10] Schmidt, M. D. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85

business intelligence use cases. These features contribute to the impression that Eureqa is the most user-friendly symbolic regression and GP system available today. Additionally, Eureqa's graph-based symbolic expression representation, combined with a highly optimized parallel C++ implementation, offers very high performance during evolutionary search, leading to high quality solutions after very short search times. The proprietary nature of Eureqa's technical platform also has disadvantages: While the Mathematica-based DataModeler and the R-based RGP automatically inherit a rich set of statistical and graphical tools for data preprocessing and result analysis from their respective platforms, Eureqa has to provide proprietary implementations, requiring users to learn a completely new and often less refined tool set. System components, such as search heuristics or search operators, cannot be modified or replaced, rendering non-trivial customizations impossible. A distinguishing feature of Eureqa is support for unsupervised learning, i.e., the automatic induction of relations in input data without explicit division into independent and dependent variables.

Others There are many other lesser known commercial and open source GP systems available today, most of which fall behind the "Pareto front" of Figure 3.1. Discipulus[11], a commercial GP system developed by RML Technologies, uses a *linear representation* in the form of machine code to encode symbolic expressions. This leads to very high performance of the evolutionary search. Unfortunately, modern algorithmic features such as multi-objective selection are largely unsupported. TinyGP is a simple open source GP system written in Java and especially geared to teaching and to be used as a baseline algorithm for GP research. The system's popularity is furthered by its use in the very popular introductory text "A Field Guide to Genetic Programming" by Poli et al. [2008].[12]

[11] see http://rmltech.com

[12] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza)

3.2 *Model Induction with RGP*

This section describes the process of data-driven model induction with RGP in detail. It should serve as a general recipe for applying RGP to common data modeling tasks. Figure 3.2 gives a course-grained overview of the RGP process in form of a flow chart. Each activity is also annotated with the participants roles. *Business experts* are the financial stakeholders of a modeling project, *domain experts* contribute application domain expertise, and *data scientists* take care of the technical modeling tasks. The RGP process is loosely based on the well-known industry standard data mining process CRISP-DM, while adding extensions specific to GP.[13]

The following subsections describe each activity shown in Figure 3.2 in more detail. Note that, while there are some minor RGP specifics, this general process could also applied to other GP systems that support typical data modeling tasks, e.g., Eureqa or DataModeler.

[13] Shearer, C. (2000). The CRISP-DM model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4):13–22

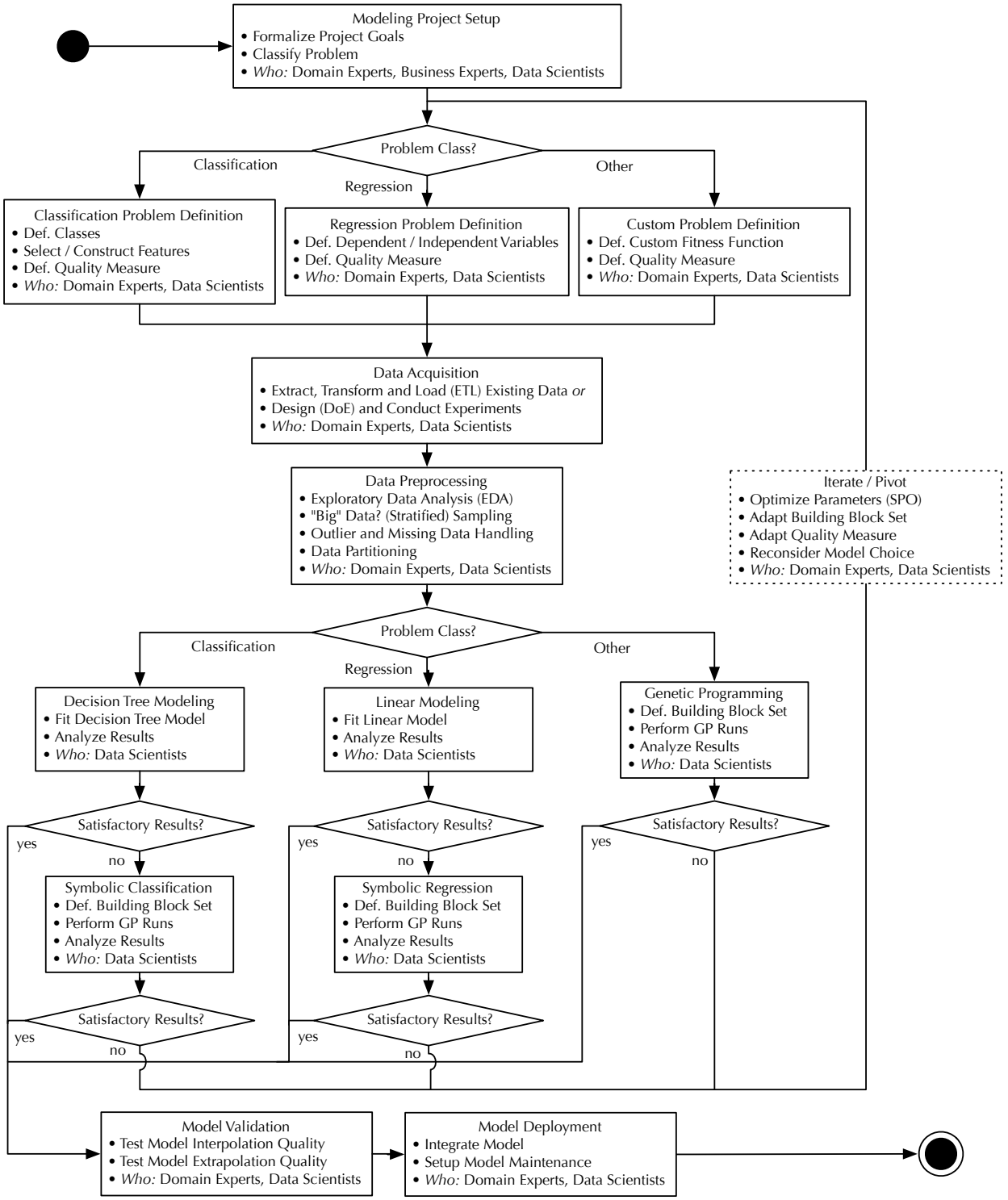


Figure 3.2: RGP model induction process. See the main text for a detailed explanation.

3.2.1 *Modeling Project Setup*

The success of any data modeling project is strongly dependent on a clear understanding of the project's goals, resources, and constraints among all stakeholders. In the modeling project setup phase, the project's goals and constraints, as well as required financial and personnel resources, including responsibilities, should be unambiguously defined. All project stakeholders are involved in the project setup phase, including business experts, domain experts, and data scientists.

Typical data modeling projects share many properties with software development projects, including high technical and organizational complexity. Furthermore, data modeling projects often a greater diversity of new technologies, as well as deeper business and domain knowledge than average software development projects. In this regard, data modeling projects often resemble research projects. Nonetheless, agile project management frameworks from the domain of software development, such as SCRUM can and should be employed to minimize project risks.[14]

[14] Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall, Upper Saddle River, NJ, 1st edition

Project management software can be very useful to enable efficient communication between project members and to track project progress. There are many commercial and open-source systems that fit this purpose. At least, they should enable a central version controlled repository for documents and data, an issue-tracking system for managing tasks and progress, and a calendering component for project planning. Redmine[15] is a web-based open-source project management software package that offers this functionality and is stable and very user-friendly.

[15] see http://www.redmine.org

As mentioned, modeling project setup is primarily a phase of information gathering. The following checklist provides a starting point on what tasks need to be completed in this phase:

- Define project goals, including measurable progress indicators and criteria for project completion.
- Define project constraints and assure the availability of required personnel, technology, and funding.
- Factor-in time and resources to account for multiple iterations, pivoting regarding methods and technologies, and project goal adjustments.
- Establish a project timetable, including milestones and deliverables.
- If possible, establish a central repository for project-related documents an data. Web-based project management software can be very helpful in this regard.
- Designate the persons responsible for project management, and for the correct and timely completion of each deliverable.
- Classify the application problem. Classes supported by RGP are classification, regression, and others (through custom fitness functions).

3.2.2 *Problem Definition*

After the application problem has been classified as either classification, regression, or other, data scientists can proceed to record problem-specific information in collaboration with domain experts. This information should be documented in detail, preferably in a central document repository accessible to all project members.

Classification In a classification problem, the target classes have to be defined. As classification as supported by RGP is a form of supervised learning, these classes have to be present in the data to be collected in the next step. If new data is to be collected, classification features have to be selected or constructed. A model quality measure has to be decided upon. R provides a rich set of state-of-the-art classification model quality measures via the ROCR package. Sing et al. [2005] describe this package in detail.[16]

[16] Sing, T., Sander, O., Beerenwinkel, N., and Lengauer, T. (2005). ROCR: Visualizing classifier performance in R. *Bioinformatics*, 21(20):3940–3941

Regression In a regression problem, independent (predictor) and dependent (response) variables have to be defined. These variables have to be present in the data to be collected in the next step. Also, a model quality measure has to be decided upon. RGP has a selection of regression model quality measures built in, including the ones described in Section 2.5.3.

Custom If the application problem at hand can neither be described as classification nor regression, a custom fitness function has to be defined. Examples for this case include the evolution of rules and strategies, e.g. for financial trading, or the evolution of support vector machine kernels.[17] See Section 2.2 for additional examples. In RGP, a custom fitness function is simply an R function mapping a GP-generated symbolic expression, also represented as an R function, to a numerical fitness value. Smaller fitness values are considered to be better.

[17] Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., and Konen, W. (2011). On the tuning and evolution of support vector kernels. Cologne Open Science, CIOP Technical Report 04/11, Bibliothek der Fachhochschule Köln, Betzdorfer Str. 2, 50679 Köln, Germany

3.2.3 *Data Acquisition*

When the application problem is defined, suitable data can be acquired. Depending on whether new data is to be recorded through controlled experiments, a formal design of experiments (DoE) phase is strongly recommended to yield high quality data. This task can be supported by most modern statistics packages. R provides comprehensive support for DoE via packages. See the CRAN task view on DoE at `http://cran.r-project.org/web/views/ExperimentalDesign.html` for details and documentation.

Existing data has to be imported into R to be used with RGP. R supports the ETL process through a comprehensive tool set. Typically, data is stored in a text file format such as comma-separated values (CSV) or XML , in a proprietary binary format such as Microsoft Excel, or in a relational database. All these formats can be read into R, either directly or with help of an add-on package.

When extracting data from a relational database, all model-relevant fields have to be present in a single table that can then be exported. This often involves *flatting* or *de-normalizing* the database before extraction. Special care must be taken when handling confidential data or individual-related data, e.g., encryption and anonymization.

After the data has been read into R and converted into a suitable R data frame, it should be stored in an standardized, non-proprietary format. R's own RDS data format is a convenient and often suitable choice, but has the drawback of being platform-specific to R. The `saveRDS` and `readRDS` functions can be used to store and load R objects to files in RDS format. A simple platform-independent alternative for medium amounts of data, i.e., R data frames of less then 2 GiB size, is to use the text-based CSV format, which can be read and written via the `read.csv` and `write.csv` functions. The *Hierarchical Data Format 5* (HDF5) is a platform-independent data format suitable for large and complex data sets that is supported by R through the RHDF5 add-on.[18]

[18] see http://www.bioconductor.org/packages/release/bioc/html/rhdf5.html

Meta-data explaining the context and meaning of the data acquired is just as important as the data itself. The following list provides a starting point on what meta-data to record:

- when, where, and by whom the data was recorded
- for what modeling purpose the data was recorded
- what format the data is represented in, including specifics like text encoding and data format version
- what software or equipment was used to record the data
- copyright and legal information pertaining to the data
- for each field in a data record:
  - title
  - short description
  - mathematical symbol, if applicable
  - data type (R supports the types `integer`, `numeric`, `logical`, `character`, `factor`, and `complex`)
  - unit of measurement
  - range, i.e., minimum and maximum value permissible
  - list of special values and their semantics (e.g. 0 or -1 encoding "no value")
- notes on any special events pertaining to the data or subsets of the data, such as unusual external influences or measuring equipment malfunctions

These information is often only easily available to domain experts, who should be closely involved in the whole data acquisition phase. All acquired data and meta-data should be stored in a central repository, together with documentation on the data acquisition process.

3.2.4 *Data Preprocessing*

In the data preprocessing phase, the acquired data is transformed into a format suitable for the modeling method and task at hand. This section focuses on data processing for data-driven GP in symbolic regression and classification tasks. As there is no universal preprocessing "recipe" suitable for all cases, the first step of preprocessing should consist of an EDA. Next, outliers and missing values should be taken care of. The data should be partitioned into training, validation, and test sets, to be able to realistically assess model quality later in the process. If necessary, the data set size should be reduced to be a good fit for GP via sampling and dimension reduction. In the case of time series data, time series embedding may be performed. As detailed domain knowledge is required throughout the preprocessing phase, domain experts should be closely involved. In the following paragraphs, the individual tasks of data preprocessing are described in more detail.

Exploratory Data Analysis In EDA, data is analyzed through visualization and descriptive statistics with the goal of forming hypotheses about potential patterns, relationships, and anomalies present in the data. Ideas developed in EDA may trigger refinements or changes to the goals of the data modeling project, or uncover the need for acquiring additional data. Tools typically used in EDA are visualization techniques such as scatter plots, histograms, QQ-plots or box plots, and the calculation of summary statistics. These visualizations can provide hypotheses about data quality, relative importance of features or independent variables on the class or dependent variable, and about suitable model building blocks, i.e., GP function sets. There are commercial EDA software packages available that focus on ease of use while providing a wide array of visualizations.[19] R also provides a large set of EDA tools including all visualizations mentioned above, through it's `plot`, `hist`, `qqnorm`, and `boxplot` functions. Summary statistics are calculated by the `summary` function. While the tools provided by R are often more flexible than commercial alternatives, they are arguably less user-friendly due to lack of a graphical user interface. For symbolic regression, RGP alleviates this problem somewhat by providing basic EDA tools as part of its web-based user interface.[20]

[19] For example, Orange (see `http://orange.biolab.si`), SAS Visual Analytics (see `http://www.sas.com/en_us/software/business-intelligence/visual-analytics.html`), and JMP (see `http://www.jmp.com`) provide mature EDA packages.

[20] see Section 3.6

Handling of Outliers and Missing Data Although symbolic regression and symbolic classification through GP are very robust modeling techniques, the quality of the resulting models can be improved by providing high-quality data. Zimek et al. [2012] provide an excellent introduction to the area of outlier detection techniques.[21] Most of these techniques are available to the R user in the form of CRAN packages. Outliers should only be removed from the data when there is clear indication, backed by domain experts, that these outliers are artifacts of data acquisition instead of being generated

[21] Zimek, A., Schubert, E., and Kriegel, H. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, 5(5):363–387

by the process to be modeled itself. As RGP in its default configuration does not support data records with missing values, these should either be repaired by infilling or omitted via the `na.omit` function, depending on the number of affected records. What infilling method to use is naturally highly dependent on the application problem and should be decided in collaboration with domain experts. Simple examples include infilling with means or medians, or with last values in the case of time series data.[22]

[22] Friese, M., Stork, J., Guerra, R. R., Bartz-Beielstein, T., Thaker, S., Flasch, O., and Zaefferer, M. (2013). UniFIeD: Univariate frequency-based imputation for time series data. Cologne Open Science, Schriftenreihe CI-plus TR 05/2013, Bibliothek der Fachhochschule Köln, Betzdorfer Str. 2, 50679 Köln, Germany

Data Partitioning In order to reliably access the quality of the generated models later in the process, data should be partitioned into training, validation, and test sets. The *training set* is the data subset used during model construction. In GP, this is the data set candidate models are evaluated against. To guard against model overfitting, an independent *validation set* is used to evaluate the performance of generated models. An additional independent *test set* is needed if multiple modeling methods are to be compared or if modeling algorithm parameters are to be tuned. In this case, models are created with each method or parameter setting based on the training set, the validation set is then used to compare their performances and select the most suitable method or parameter setting. Finally, the test set is used to assess the performance of the model created by that method or parameter setting. There are techniques that give more dependable indications of model performance, such as cross validation. These are seldom used for compute time intensive modeling methods such as GP, though. R's `sample` function can be used to randomly partition data sets. In the case of time series data, the data should be divided into consecutive coherent subsets. RGP's web-based user interface also provides graphical tools for data partitioning.

Size Adjustment Data-driven GP is a compute time intensive modeling method, where much compute time is spend evaluating candidate models, i.e., applying candidate models to each data record in the training set. For this reason, the training set should be as small as possible, while still containing enough information to characterize the process to be modeled with sufficient fidelity. There is no universal method of determining how much data is enough to induce high-quality models from, as many factors determining the answer to this questions depend on the process to be modeled. As a coarse rule of thumb, the number of records r in the training data set should be roughly related to the number of relevant fields or true dimensionality d of the data by $r :\approx s^d$, where s is the number of records per dimension. This number has then to be adjusted with regard to data quality, as the presence of noise and measurement errors can be countered by higher data volume, at least to a certain degree. The value of r puts a theoretical limit on what model structures can be fit to the data exactly. Also note that effects commonly described by the term "curse of dimensionality"

make it extremely difficult to build non-overfitted models for high dimensional data. When looking at polynomials, a class of model structures often included in the search space of symbolic regression, $s - 1$ is the degree of the polynomial that fits the data exactly. In practice, the value for r (or s) can be chosen as large as the available compute time budget allows, with adjustments based on intuition gained during EDA. GP with multi-objective selection based on model error and model complexity, as implemented in RGP, is robust to the presence of spurious dimensions, i.e., fields that have no functional relationship to the class or dependent variable. In fact, GP has been successfully applied to selecting relevant dimensions in high-dimensional data. When working with time series data, RGP's `embedDataFrame` function can be applied at this stage of preprocessing to transform a time series regression problem to a time-independent symbolic regression problem.

3.2.5 *Classical Modeling*

When working on a classification or regression problem, it is highly advisable to apply classical modeling techniques before turning to the more advanced and also more complex method of symbolic classification or symbolic regression. When working with RGP, applying classical modeling approaches is particularly easy as at this stage, the data set is already available in R, and R provides mature and flexible implementations for most classical statistical modeling techniques. Adler [2010] concisely introduces relevant classical statistical models for classification and regression.[23] Evaluating the classical modeling techniques enumerated in the next two paragraphs has clear advantages over the alternative of directly turning to more advanced GP-based approaches: 1) The techniques are very mature and well understood in theory and practice. 2) Mature analysis methods exist for these models and model understandability is good. 3) Compute time requirements are very low compared to GP. 4) The resulting models offer good understandability, given that maximum model complexity is limited through suitable parameter settings. 5) Even when result accuracy is not sufficient, a baseline result useful for comparisons is created.

[23] Adler, J. (2010). *R in a nutshell*. O'Reilly, Sebastopol, CA

Classification When working on a classification problem, at least basic recursive partitioning techniques available through R's `rpart` package should be evaluated.[24] This techniques create decision tree models that can be easily deployed. Visualized as trees, these models offer good understandability and insight into the application problem structure.

[24] see `http://cran.r-project.org/web/packages/rpart`

Regression In the case of regression problems, at least basic linear regression models available through R's `lm` function should be evaluated, unless it is certain that the process to be modeled is clearly non-linear. Linear models for data of medium dimensionality usu-

ally offer good understandability and are easy to deploy. In the case of time series data, autoregressive moving average model should be evaluated. These models are linear in the sense that the predicted values are a linear function of current and past values of the same time series and current and past values of a series of random variables.

3.2.6 *Genetic Programming*

In the case that traditional classification or regression models prove to be insufficient or in the case of a custom application problem, GP runs should be setup and performed. In the case of a regression or classification problem, RGP's `symbolicRegression` function can be used for this purpose. This function provides an interface similar to R's `lm` function. If a custom fitness function is needed, RGP's `geneticProgramming` function should be used instead. These functions have a large number of parameters, including parameters for defining the GP function and constant sets, and parameters to define the search heuristics to be used. All parameters have sensible default values that adapt to the problem class at hand. After GP run completion, a set of models of suitable quality and complexity is selected from the model Pareto front for further analysis in the model validation phase. Section 3.3 illustrates this step by means of examples, while Section 3.5 provides a detailed overview of all modeling features. See RGP's online documentation on `symbolicRegression` and `geneticProgramming` for additional reference documentation.

3.2.7 *Model Validation*

In model validation, one or more models are analyzed to assess their performance under realistic conditions and to gain further understanding of the process to be modeled. Models are evaluated on test data and performance measures such as MAE and R^2 (if applicable) are calculated. Depending on the intended use, not only model interpolation capability, but also model extrapolation capability is tested. When testing extrapolation capability, a model is applied to a region of the input data space not present in the training data set. This work uses a bounding box definition of extrapolation, i.e., data points outside of the rectangular hull of the training data points are in the extrapolative region.[25] As both GP-generated models and models created by the classical approaches outlined previously are white-box models, model validation should also include discussing the models with domain experts. To simplify the presentation of models to non-technical users, RGP contains tools for visualizing models as trees or as formulas in mathematical notation. Also included are tools to analyze the input variables, i.e., features or independent variables, used by a set of models. Domain experts should ensure that the variable importance reported by these tools matches their intuition of the modeled process.

[25] There are other definitions of extrapolation, see Ebert et al. [2014] for an overview.

Ebert, T., Belz, J., and Nelles, O. (2014). Detektion von Extrapolation. *Proceedings of the 24. Workshop on Computational Intelligence, Dortmund, Germany*, 50:17–32

3.2.8 Model Deployment

If the model validation phase found a model or set of models fit for practical use, the model or models can be deployed. This step is highly dependent on the intended use for the model. In the simplest case, models are only used for gaining understanding of a data set and it's underlying process or for one-shot predictions, in which case the model deployment step can be omitted. If a model is to be used more regularly, it has to be documented. This documentation should be made available in a central repository accessible by all project members, including the model itself. Model documentation should include the following:

- for what modeling purpose the model was created
- when, where, and by whom the model was created
- what method, including all parameter settings, was used to create the model
- what software, including version, was used to create the model
- model quality measures as determined in model validation
- insights on the modeled process derived from the model
- notes on any non-trivial assumptions or constraints of the model

In the case the model is to be deployed as part of a software system, it has to be integrated. Standard software engineering practice, including integration tests, should be followed in this step. White-box models, e.g. symbolic expressions generated by GP, are generally easier and more cost-effective to deploy then black-box models that require specialized software components for model execution. The last step in model deployment should be the setup of a model maintenance process. Goal of this process is to monitor and control prediction quality during the active use of a model and to warn users if the underlying process changed in such a way that an existing model does no longer fit. Methods from the field of statistical process control, such as control charts, can be used to monitor model prediction error.[26]

[26] Oakland, J. (2003). *Statistical Process Control*. Quality management Series. Butterworth-Heinemann, Newton, MA

3.2.9 Iteration and Pivoting

Data modeling as a whole, as well as each of its phases, are iterative processes. In all phases, data scientists and domain experts are bound to learn new facts about the data, the underlying process to be modeled, and the suitability of the modeling techniques and parameter settings used. In order to make this process as efficient as possible, each step in each iteration should be documented. Model parameter settings should only be changed based on clearly documented indications, such as knowledge gained from EDA or earlier modeling runs. If model parameter settings are to be explored, this exploration should be based on a DoE to obtain optimal results for the time invested. See Chapter 5 for a detailed study of how to apply DoE to GP. When using GP, the building block set, i.e., the set of functions and constants, should be adapted based on knowledge gained in previous iterations. Also, the choice of model

quality measure used during evolution can have a strong effect on the results of a GP run and should be reconsidered in the case of model quality issues. Another option to increase model quality is to increase the allowed compute time budget. In the case that still no model of satisfactory quality is found by GP, black-box models should be reconsidered, as they might be able to attain higher model accuracy at the expense of model understandability.

The process of model induction with RGP is illustrated through basic example applications in the next sections. Chapter 6 provides additional real-world application examples.

3.3 RGP Tutorial Examples

To help getting started with RGP, this section provides a set of tutorials, starting with simple tasks including getting RGP up and running in an existing R installation, up to more advanced topics like strongly typed GP. All tutorials are meant to be followed step-by-step, in a running R session.

In the reminder of this chapter, the following typographical conventions are used when displaying R interpreter input and output:

```
Input entered by the user is marked by a thick line to the left.
```

```
Output returned by RGP is marked by a thick line to the right.
```

All tutorial examples are based on RGP release 0.4-0, the current version of RGP at the time of this writing.

3.3.1 Tutorial I: RGP Installation

RGP is available as an R package on the comprehensive R archive network (CRAN), making installation very simple. R is available for all common operating system platforms at http://www.r-project.org. Follow the instructions on this website to install R.

Then, to install RGP and all it's dependencies, issue the following command in a running R session:

```
install.packages("rgp")
```

A prompt asking to select a CRAN mirror will appear if this is the first time an R package is installed in the running R session. Select a mirror location near you. The installation of RGP may take some time, as dependencies are downloaded and compilation steps are performed.

To install the optional web-based user interface for RGP, issue the following command:

```
install.packages("rgpui")
```

Compiler warnings will be reported on some operating system platforms with some configurations and can usually be safely ignored. In case of installation errors, support is available through the RGP website at http://rsymbolic.org/projects/rgp.

3.3.2 *Tutorial II: Basic Genetic Programming*

This tutorial provides an interactive walkthrough of solving a basic artificial symbolic regression problem with GP. Only low-level RGP functionality is used, high-level convenience functions are intentionally avoided to make each step in the modeling process clear and explicit.

Modeling Project Setup In this first example, RGP will be configured to create polynomial approximations of the sine function. To make RGP's functionality available in a running R session, the package has to be loaded via R's `library` command:

```
library("rgp")
```

Problem Definition In RGP, candidate solutions are represented as regular R functions. The bodies of these functions are build from a set input variables, a set of constants, and a set of function symbols. The members of these sets are often referred to as GP building blocks. In other words, these three sets define the symbolic expression search space.

As the task of this example is the approximation of the sine function through polynomials, a function symbol set containing only addition, multiplication, and subtraction is defined:

```
functionSet1 ← functionSet("+", "*", "-")
```

Then, a set of input variables containing just the symbol x is defined. Thereby the search space is restricted to univariate functions, i.e., function of one variable:

```
inputVariableSet1 ← inputVariableSet("x")
```

Finally, the set of constants is defined. Constants are not created directly, but via constant factory functions. Each time a constant has to be created during GP search, RGP will call a constant factory function. Here, a single constant factory that returns constants from a normal distribution is defined:

```
constantFactorySet1 ← constantFactorySet(function() rnorm(1))
```

The fitness function, or objective function, associates a numerical fitness value to a candidate solution. RGP relies on the fitness function to guide evolutionary search. The fitness function defines the problem to be solved by GP. In this example, RGP is employed to find functions approximating the sine function in the interval `interval1` $[-\pi, \pi]$. This interval is sampled in steps of size 0.1:

```
interval1 ← seq(from = -pi, to = pi, by = 0.1)
fitnessFunction1 ← function(f) rmse(f(interval1), sin(interval1))
```

By default, RGP minimizes fitness values, so lower values should be associated with better solutions. Here, the RMSE of a given sine approximation against the true sine function is used as a fitness function.[27]

[27] The problem defined here is a typical symbolic regression problem. RGP also features a simple interface for symbolic regression, which is introduced in the next tutorial.

Data Acquisition and Data Preprocessing As this example uses a custom fitness function, no explicit data acquisition or preprocessing steps are necessary.

Genetic Programming RGP is now configured and ready to start an evolutionary search for symbolic expressions of good fitness values:

```
set.seed(1)
gpResult1 ← geneticProgramming(
  functionSet = functionSet1,
  inputVariables = inputVariableSet1,
  constantSet = constantFactorySet1,
  fitnessFunction = fitnessFunction1,
  stopCondition = makeTimeStopCondition(5 * 60))
```

The first command will set R's random number generator seed to a defined value (here: 1) in order to create reproducible results. Then, a GP run that stops after five minutes is performed. The results of this run are stored in the R variable `gpResult1`. The GP runtime budget can be adjusted by changing the parameter to `makeTimeStopCondition` to another number of seconds.

Finally, the best sine approximation found during the GP run is selected:

```
bestSolution1 ← gpResult1$population[[which.min(
  gpResult1$fitnessValues)]]
```

As RGP represents GP individuals as R functions, the solution can be directly printed:

```
bestSolution1
```

This command should create the following output:[28]

[28] Depending on the RGP version used, this output might differ.

```
function (x)
(x - (-0.636016463701753 * (1.38918749561857 + (x + x +
  x + (x + (x + 0.135504518362805))) * (x - x *
  1.02940921798835)) + x)) * (x + ((0.366442227806443 -
  x) * x + (0.315373883934325 + (x * 1.19762981213713 *
  (-3.3644961867303 * x) + -0.0888751683194937))) *
  (x * 0.00529237751006351))
```

For deployment, this expression can be readily translated to mathematical notation or to source code in most programming languages.

Model Validation and Model Deployment A plot of the polynomial approximation *bestSolution*1 versus the true sine function is cre-

ated:

```
plot(y = bestSolution1(interval1), x = interval1,
  type = "l", lty = 1, xlab = "x", ylab = "y")
lines(y = sin(interval1), x = interval1, lty = 2)
```

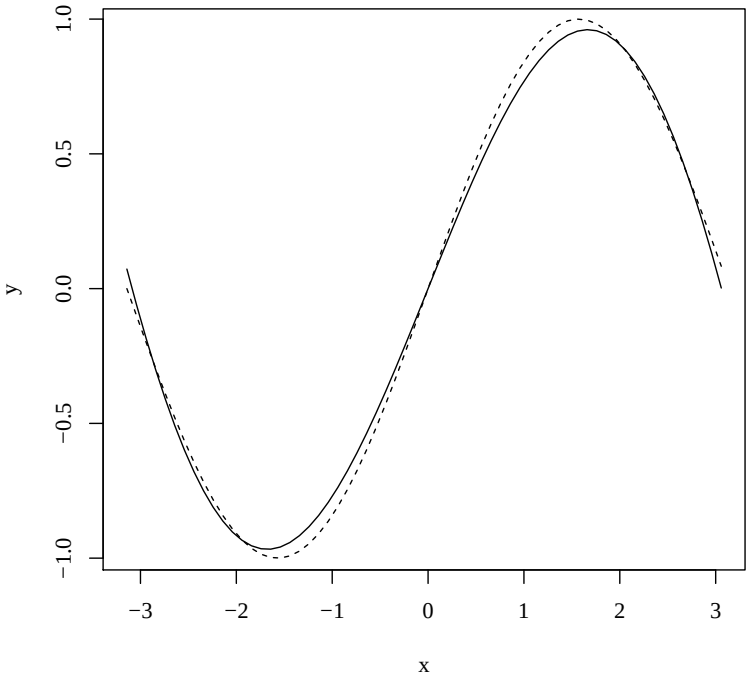


Figure 3.3: Best GP-generated polynomial approximation (solid line) versus true sine function (dashed line).

Figure 3.3 shows the plot created by these commands. This simple example is well suited to demonstrate the importance of model validation. Here, the model's extrapolation quality is checked visually by plotting an extended interval:

```
interval2 ← seq(from = -1.5*pi, to = 1.5*pi, by = 0.1)
plot(y = bestSolution1(interval2), x = interval2,
  type = "l", lty = 1, xlab = "x", ylab = "y")
lines(y = sin(interval2), x = interval2, lty = 2)
```

The resulting plot, shown in Figure 3.4, clearly shows how model accuracy deteriorates in an extrapolative setting. Here, this deterioration has been caused intentionally by restricting the space of possible models to polynomials. This effect can occur even when using less restricted function sets, therefore models should be carefully validated in both interpolative and extrapolative settings.

Next Steps This concludes this basic tutorial. Of course there is much room for experimentation. For example, the members of `functionSet1` could be extended by adding the cosine function `cos`. Note that many RGP convenience functions have been omitted that

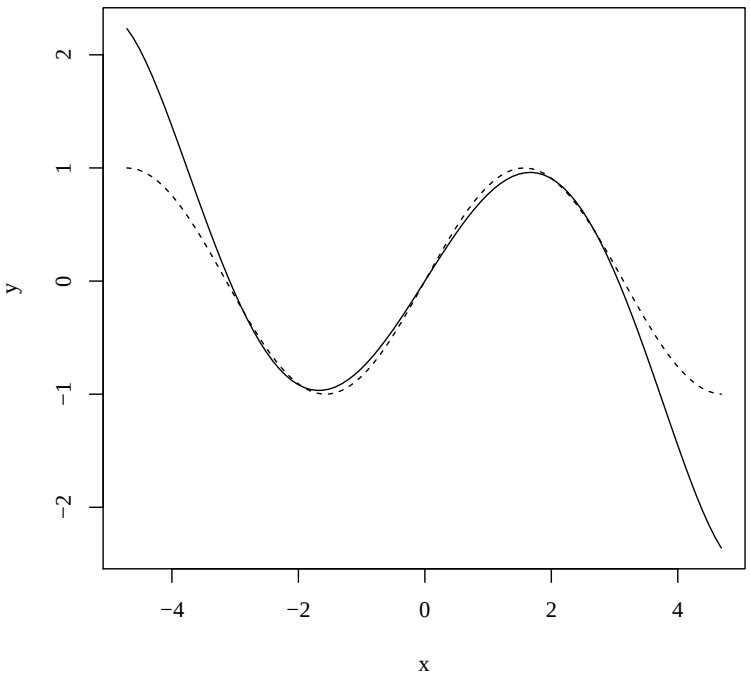


Figure 3.4: Best GP-generated polynomial approximation (solid line) versus true sine function (dashed line), plotted in an extended interval.

would have made this particular example much shorter.

3.3.3 *Tutorial III: Symbolic Regression of a Damped Pendulum*

RGP offers convenience functions to simplify the solution of common tasks in GP. This tutorial shows how to use the `symbolicRegression` function to solve symbolic modeling and regression tasks with only minimal configuration work.

Modeling Project Setup Theme of this tutorial is the discovery of a mathematical formula describing the behavior of a physical system based on measurement data, i.e., symbolic regression. For sake of simplicity and clarity, this data will be generated by applying a text-book formula describing a damped pendulum. The task of RGP then becomes the rediscovery of an equivalent formula and of the numerical values of the formula's parameters.

Problem Definition The formula below, given as a *factory function* in R, represents a damped pendulum. The arguments are the starting amplitude A_0, gravity g, pendulum length l, phase ϕ (phi), damping factor γ (gamma), and radial frequency ω (omega). Figure 3.5 illustrates some of these arguments.

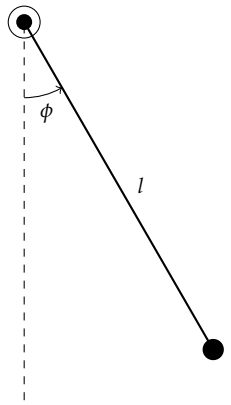


Figure 3.5: Illustration of some of the arguments to the model equation for a damped pendulum.

```
makeDampedPendulum ← function(A0 = 1, g = 9.81, l = 0.1,
  phi = pi, gamma = 0.5) {
  omega ← sqrt(g/l)
  function(t) A0 * exp(-gamma * t) * cos(omega * t + phi)
}
```

A factory function is a function that returns another function configured according to the factory functions argument. The factory function `makeDampedPendulum` can be applied to generate functions describing the deflection of concrete pendulums of different specifications at a certain point in time t:

```
pendulum1 ← makeDampedPendulum(l = 0.5)
pendulum2 ← makeDampedPendulum(l = 1.2, A0 = 0.5)
```

To illustrate, the deflection of these pendulums can easily be plotted against time (see Figure 3.6). In this plot, `pendulum1` is shown as a solid line, and `pendulum2` is shown as a dashed line.

```
interval1 ← seq(from = 0, to = 10, by = 0.05)
plot(y = pendulum1(interval1), x = interval1,
  type = "l", lty = 1, xlab = "t", ylab = "deflection")
lines(y = pendulum2(interval1), x = interval1, lty = 2)
```

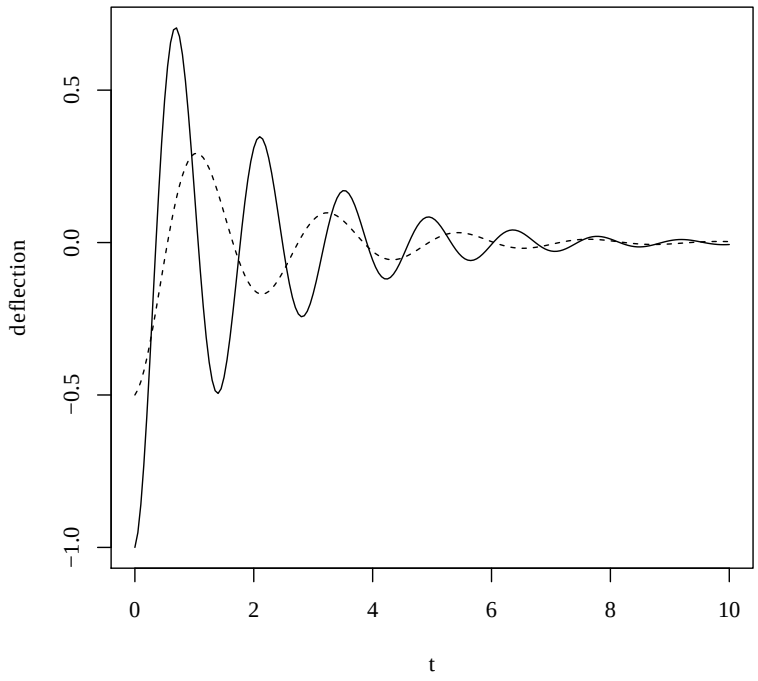


Figure 3.6: Deflection against time of two example pendulums. `pendulum1` is shown as a solid line, `pendulum2` is shown as a dashed line.

Data Acquisition and Data Preprocessing A data frame of 512 samples of `pendulum1` in the time interval $[1, 10]$ is created. To simulate the imperfections of real measurement data, normally distributed

noise with mean 0 and standard deviation 0.01 is added to the simulated values:

```
xs1 ← seq(from = 1, to = 10, length.out = 512)
pendulum1Data ← data.frame(
  time = xs1,
  deflection = pendulum1(xs1) + rnorm(length(xs1), sd = 0.01))
```

A plot of this data is generated for visual inspection (see Figure 3.7). Each of the 512 data points is represented by a solid dot in this figure. The influence of the normally distributed noise introduced to simulate the imperfections of real-world measurements can be clearly seen.

```
plot(pendulum1Data, xlab = "t", ylab = "deflection",
  pch = 20, cex = 0.5)
```

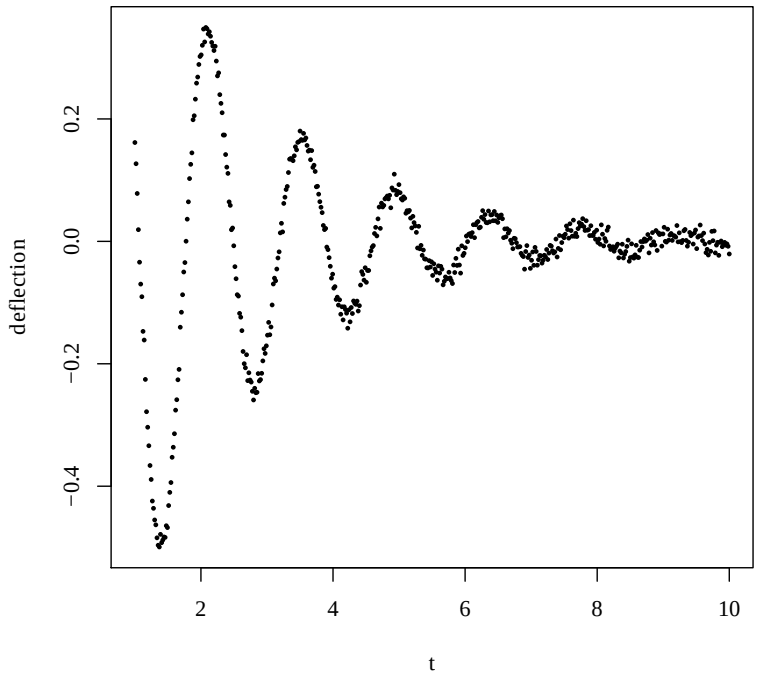


Figure 3.7: The generated data frame `pendulum1Data`. Each of the 512 data points is represented by a solid dot. Note the noise introduced to simulate the imperfections of real-world measurements.

Classical Modeling In a real-world setting, where the properties of the underlying process generating the data are not known, modeling techniques that are simpler than symbolic regression should be applied first. Here, as an example, a linear model is fitted to the data and saved in the variable `linearModel1`. Then, a summary of the model is requested:

```
linearModel1 ← lm(deflection ~ time, data = pendulum1Data)
summary(linearModel1)
```

This summary should be similar to the example reproduced here:

```
Call:
lm(formula = deflection ~ time, data = pendulum1Data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.44877 -0.03546 -0.00913  0.04235  0.39366

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.062603   0.014030  -4.462    1e-05 ***
time         0.008848   0.002306   3.837  0.00014 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error:0.1358 on 510 degrees of freedom
Multiple R-squared: 0.02806, Adjusted R-squared: 0.02616
F-statistic: 14.73 on 1 and 510 DF,  p-value: 0.0001399
```

The low adjusted R^2 value of 0.02616 already indicates that a linear model is not a good fit for the data. Of course, this had to be expected considering the non-linear nature of the damped pendulum used in this example.

Genetic Programming As linear modeling does not produce satisfactory results, a symbolic regression is started. At this point, the RGP package should have been loaded into the running R session via the command `library("rgp")`.

A time limit of 15 CPU-core minutes is chosen for the symbolic regression run. All other parameters are kept at their default values. RGP's `symbolicRegression` function mimics R's `lm` function to make it easily approachable to users accustomed to linear modeling with R:

```
srResult1 ← symbolicRegression(deflection ~ time,
  data = pendulum1Data,
  stopCondition = makeTimeStopCondition(15 * 60))
```

Model Validation and Model Deployment Selection and plotting of the model with best fitness can be performed as follows:

```
bestModel1 ← srResult1$population[[which.min(
  srResult1$fitnessValues)]]
plot(y = bestModel1(xs1), x = xs1, type = "l",
  lty = 1, xlab = "x", ylab = "y")
lines(y = pendulum1(xs1), x = xs1, lty = 2)
```

A slightly improved version of the output produced by these commands is shown in Figure 3.8. In this plot, the true pendulum without noise is shown as a solid line, the solution found by RGP based on noisy data is shown as a dashed line. Also, the true formula and the symbolic regression result are shown in the plot's

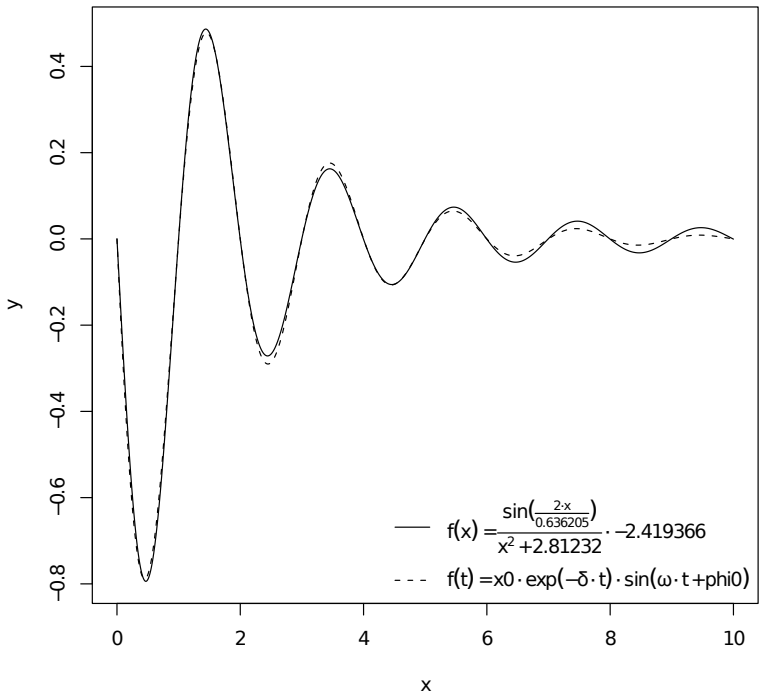


Figure 3.8: Example plot of the result of a GP run for symbolic regression of a damped pendulum. The true pendulum without noise is shown as a solid line, the solution found by RGP based on noisy data is shown as a dashed line. Also, the true formula and the symbolic regression result are shown in the legend to the lower right. Note that the result may differ depending on random seed, compute time budget, and RGP release.

legend to the lower right. Note that the exact result may differ depending on random seed, compute time budget, and RGP release. In this example, RGP did not find an exact solution, although the difference is minor. The exponential damping term in the true formula has been approximated by a polynomial damping term. While the true formula includes symbolic constants, the solution found by RGP include approximated constants found during evolutionary search.

As in the previous tutorial example, the solution should be validated in an extrapolative setting. As the procedure is exactly the same, this step is omitted here for brevity. Model deployment can also proceed as indicated in the last tutorial example.

Next Steps RGP's `symbolicRegression` command offers many configuration options to explore. See the online help available by typing `?symbolicRegression` on an R command line for details. Perhaps most importantly, `symbolicRegression` supports multivariate regression simply via R's formula interface. For example, to perform symbolic regression in two variables x_1 and x_2 with output variable y, the formula `y ~ x1 + x2` would be used as the first argument to `symbolicRegression`.

3.3.4 *Tutorial IV: Strongly Typed GP*

The theme of this tutorial example is the evolution of boolean functions by means of strongly typed genetic programming. Strongly typed GP is more complex than untyped GP as used in the previous tutorials, but allows RGP to be applied to a much broader set of tasks.

Modeling Project Setup In this tutorial example, strongly typed GP will be applied to discover symbolic representations of the 3-parity function. The 3-parity function is the parity function for 3 bits, i.e., a function with three input parameters. For this demonstration, first a R-implementation of the general parity function is defined:

```
parity ← function(x) {
  numberOfOnes ← sum(sapply(x, function(bit) if (bit) 1 else 0))
  numberOfOnes %% 2 != 0
}
```

For a boolean input vector x, the parity function returns true if the number of ones in x is odd. This general function is then specialized to the case of three parameters (3-parity) by means of the following wrapper function:

```
parity3 ← function(x1, x2, x3) parity(c(x1, x2, x3))
```

Problem Definition At this point at the latest, the RGP package should be loaded by issuing the command `library("rgp")`. Next, RGP's tool function `makeBooleanFitnessFunction` is applied in order to convert `parity3` to a fitness function. The resulting `parityFitnessFunction` has one parameter of type "boolean function". It returns the number of different places in the value table of a boolean function presented as a parameter and the value table of the `parity3` function:

```
parityFitnessFunction ← makeBooleanFitnessFunction(parity3)
```

This fitness function represents a distance metric (a norm): The Hamming distance of a 3-parameter boolean function, given as the fitness function's parameter, to the `parity3` function. In other words, the `parityFitnessFunction` returns the number of input vectors for which a given boolean functions differs in output from the `parity3` function. As there are $2^3 = 8$ different possible boolean input vectors of length 3, the worst fitness is 8, and the best fitness is 0. Note that, due to the structure of the search space, solutions with absolute worst fitness (8) can be transformed to solutions with perfect fitness (0) by simply negating their output. In presence of a suitable mutation operator, this transformation can occur in a single step. GP search spaces of rich and interesting structure are very common, and it is often beneficial to customize GP search heuristics, variation operators, or solution representations to exploit

existing knowledge on search space structure to speed up search considerably.

Data Acquisition and Data Preprocessing In practice, the true boolean function would of course be unknown. Instead, a data set representing (a subset of) the value table of the boolean function sought would be available. This task, i.e., the task of evolving boolean functions based on fitness functions that qualify as hamming norms, is a common GP task with many practical applications. Much research dealing with GP solution representations and variation operators optimized for this task has been done. For example, Wiesmann [2001] introduces the use of ordered binary decision diagrams as GP solution representations of binary functions.[29]

In this artificial example, the true boolean function is known. Therefore, no data acquisition or data preprocessing steps are necessary.

[29] Wiesmann, D. (2001). Anwendungsorientierter Entwurf evolutionärer Algorithmen. Dissertation, Universität Dortmund, Germany

Genetic Programming With the custom fitness function defined, the next step is to define the set of symbolic expressions to be searched by RGP. This is done by providing GP building blocks for boolean functions. The constant factory set contains a single constant factory that creates boolean constants by fair coin-tosses:

```
booleanConstantFactory ← function() runif(1) > .5
booleanConstantSet ← constantFactorySet(
  "booleanConstantFactory" %::% (list() %->% st("logical")))
```

The function set contains the boolean functions *and* (`&`), *or* (`|`) and *not* (`!`):

```
booleanFunctionSet ← functionSet(
  "&" %::% (list(st("logical"), st("logical")) %->% st("logical")),
  "|" %::% (list(st("logical"), st("logical")) %->% st("logical")),
  "!" %::% (list(st("logical")) %->% st("logical")))
```

The input variable set contains the three function parameters `x1`, `x2`, and `x3`:

```
booleanInputVariableSet ← inputVariableSet(
  "x1" %::% st("logical"),
  "x2" %::% st("logical"),
  "x3" %::% st("logical"))
```

The building block definitions above use a special RGP syntax for type annotations. The *`expression`* `%::%` *`type`* operator associates an R expression with an RGP type. An RGP type is either a base type of the form `st(`*`type name`*`)` or a function type of the form `list(`*`parameter type 1`*`, `*`parameter type 2`*`, ...)  %->%` *`result type`*. [30] This is a recursive definition, meaning that RGP types can express types for higher order functions, which makes them quite flexible. The theoretical basis of RGP's type system is the *simply typed lambda calculus*.[31] A noteworthy limitation of this system is the lack of the generic types available in many modern strongly typed

[30] The base type constructor is called `st` as an abbreviation for "S-Type". The S language was the predecessor of the R language.

[31] Barendregt, H., Abramsky, S., Gabbay, D. M., Maibaum, T. S. E., and Barendregt, H. P. (1992). Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309, New York. Oxford University Press

programming languages such as C++ or Java.

With these definitions, we are able to explain the semantics of the types associated with the GP building blocks above:

- `list() %->% st("logical")` is the type of a function with no arguments that returns a boolean value.[32] This is the type of the single constant factory in the `booleanConstantSet` defined above.
- `list(st("logical"), st("logical")) %->% st("logical")` is the type of a function taking two boolean values as arguments and returning a boolean value. This is the type of each function in the `booleanFunctionSet` defined above.
- Trivially, `st("logical")` is the type of boolean values. This is the type of each input variable in the `booleanInputVariableSet` defined above.

[32] a `logical` in R's terminology

With the fitness function and search space defined, all is ready to start a strongly typed GP run:

```
typedGpResult1 ← typedGeneticProgramming(parityFitnessFunction,
  type = st("logical"),
  functionSet = booleanFunctionSet,
  inputVariables = booleanInputVariableSet,
  constantSet = booleanConstantSet,
  stopCondition = makeTimeStopCondition(30))
```

Note that `typedGeneticProgramming` expects the range type of the solution functions that are to be generated as a second parameter, as this type information is not contained in the building block definition. Because boolean functions are to be generated, `st("logical")` is used as the argument's value here.

After running for 30 CPU-core seconds, the result of the GP run is assigned to the variable `typedGpResult1`. As in the previous tutorials, the runtime (in seconds) can be adjusted by changing the parameter to `makeTimeStopCondition`.

Model Validation and Model Deployment Selection of the boolean function with best fitness is performed much like in the previous tutorials:

```
bestFunction1 ← typedGpResult1$population[[which.min(
  typedGpResult1$fitnessValues)]]
bestFunction1
```

The second line will print the solution with best training fitness. As in the previous example, the exact results may differ depending on random seed, compute time budget, and RGP version used:

```
function (x1, x2, x3)
x2 & x1 & x3 | !(x3 | (!x2 | x1)) | x3 & (!x1 & !x2)
```

See RGP's online documentation for details on visualizing and analyzing typed GP results. For example, the Rrules package, available on CRAN as an extension to RGP, can be employed for rule-based simplification of boolean functions.

3.4 RGP Architecture

This section discusses the architecture and design of RGP, both from a user perspective and from a software engineering perspective. As RGP is based on R and adheres to R's design principles as much as possible, this section will also include a discussion of the design principles underlying the R language and ecosystem as a whole.

3.4.1 Design Considerations

Modern GP systems, as introduced in Section 3.1, are complex aggregates of algorithms for solving not only GP specific tasks, such as solution creation, variation, and evaluation, but also more general EC tasks, like single- and multi-objective selection, and even general data analysis and statistics tasks like design of experiments, data preprocessing, result analysis and visualization. Packages like MATLAB, Mathematica, and R already provide solutions for most of these general tasks, greatly simplifying the development of a GP system based on one of these environments, while also lowering the barrier of entry for users already familiar with the software environment used.

RGP is based on the R environment for several reasons. First, there seems to be a trend towards employing statistical methods in the design and analysis of evolutionary algorithms, including modern GP variants.[33] Second, R 's open development model has led to the free availability of R packages for most methods from statistics and many methods from EC. Also, the free availability of R itself makes RGP accessible to a wide audience. Third, the R language supports *computing on the language*, which greatly simplifies symbolic computation inherent in most GP operations. In addition, parallel execution of long-running GP experiments is adequately supported by R's parallel computation functionality.

[33] Sun, Y., Wierstra, D., Schaul, T., and Schmidhuber, J. (2009). Efficient natural evolution strategies. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCo 2009)*, pages 539–546, New York. ACM Press; and Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2009). F-race and iterated F-race: An overview. In Bartz-Beielstein, T. et al., editors, *Empirical Methods for the Analysis of Optimization Algorithms*. Springer, Berlin

The following list summarizes the high-level design goals that motivated RGP's architecture. The design goals are loosely ordered by priority:

- *Effectiveness:* RGP should be effective in real-world applications. For regression problems, RGP should be able to generate models that are of comparable accuracy with models generated by other regression techniques from CI, machine learning, and statistics.
- *Approachability:* RGP should be easy to understand for users without a background in programming or computer science. The target audience includes statisticians, data scientists, data analysts, and engineers.
- *Extensibility:* RGP should be easily extensible to new GP application classes. Support for GP application classes of high practical relevance should be built-in.
- *Modularity:* Within limits of reasonable efficiency, each RGP function should be usable on its own. There should be no de-

pendencies on RGP specific infrastructure, no proprietary data structures, no side-effects, and no conventions in addition to those valid for all R functions.
- *Efficiency:* RGP should be efficient enough for real-world application. When efficiency conflicts another design goal, a compromise should be sought, where efficiency should not be of highest priority. Each RGP function should be user-replaceable by a less-generic but more efficient variant (specialization).

In the following, some examples of how individual architectural features help to realize RGP's high-level design goals are given. More details on RGP's feature set will be provided in Section 3.5. A more technical account on the software engineering side of RGP's design will be given in Section 3.4.2.

Effectiveness In Chapters 5 and 6 it will become clear that the level of GP system effectiveness for real-world applications is largely a function of the search heuristic, GP function set, and GP parameter settings used. The design features to facilitate extensibility and modularity to be introduced in the following paragraphs help in this regard, as they enable quick adaption to new real-world problem classes. In order to find high quality solutions within reasonable time limits, a GP system also needs a reasonably efficient implementation. Architectural features of RGP helping efficiency are also a topic of the next paragraphs.

Approachability The high-degree of integration into the R environment should make RGP very approachable to existing R users. As demonstrated in Section 3.3.3, the common GP task of symbolic regression is supported through a simplified workflow resembling the linear modeling workflow well known to most R users. This includes the specification of symbolic regression models through R's formula syntax and the support for generating predictions from RGP models through R's generic function `predict`. With the help of RGP's type system, this workflow can also be applied to classification problems. The design of the R language itself is geared towards interactive use by non-expert users. Its relatively high approachability is in part manifested through its high popularity with end users. Additionally, the web-based graphical user interface, described in Section 3.6, should make RGP approachable even to users not familiar with R or to users not at ease with command-line user interfaces in general. Finally, comprehensive documentation for each RGP function is available via R's online help system.

Extensibility Much like R, RGP's design broadly follows the functional programming paradigm, with some exceptions to aid efficiency. The system consists of a relatively fine-grained set of task-specific, generally side-effect-free, R functions. Each function implements a clearly defined small piece of functionality, such as a

specific error measure or a specific mutation operator. With few exceptions, each function is simple enough to fit on a single screen page. There is no global configuration infrastructure, i.e., no global configuration file or in-memory configuration data structure. All configuration is provided through function arguments, which may be functions. For example, the crossover operator is provided as a function-valued argument to a function implementing the GP search heuristic to be used in a GP run. RGP keeps the use of proprietary data structures at a reasonable minimum. Whenever possible, standard R data structures are used. When additional context information is needed, it is attached to standard data structures via R's attribute mechanism. For example, a population of GP individuals is represented by a vector of R functions optionally annotated by type information. Combined, these design features greatly facilitate extensibility. To provide additional functionality, adding a R function suffices in most cases. This function then can be passed as a parameter to existing RGP functions. No registration of the new functionality or adaption of configuration infrastructure is necessary. As there are nearly no side-effects, all dependencies of each RGP function are explicitly stated in the function's arguments.[34]

[34] Exceptions to this rule, e.g. functions used for logging and plotting, are stated in the online documentation of the functions affected.

Modularity The design features enumerated in the previous paragraph on extensibility also contribute to RGP's modularity. The use of data structures proprietary to RGP is kept at an absolute minimum, while standard R data structures are used whenever reasonably possible. One the one hand, this allows RGP functions to be used in new contexts. On the other hand, and perhaps of even higher importance for most RGP users, it allows other functions from the large ecosystem of R packages to be used with RGP. For example, R's `deriv` function that computes symbolic derivatives can be used on RGP solution expressions directly, without any conversion. As a consequence of this design decision, R's objective-oriented features are nearly unused in RGP, the only exception being the definition of methods for some built-in generic functions, such as for plotting or model prediction.

Efficiency At the time of writing, efficiency of interpreted R code is unfortunately relatively poor.[35] This situation can be alleviated to some degree by using R's vector operations instead of loops as much as possible, which RGP does. RGP adopts the strategy of function specialization to reach an efficiency level suitable for real-world application. For most performance-critical functions, including GP operators for initialization, mutation, recombination, and evaluation, less flexible but higher performing variants exist. These variants are implemented in the lower level C programming language and are tuned for performance. When using one of RGP's application specific interfaces for symbolic regression or classification, these faster variants are used by default. Except from having fewer configuration options and higher performance, these variants

[35] Morandat, F., Hill, B., Osvald, L., and Vitek, J. (2012). Evaluating the design of the R language - Objects and functions for data analysis. In Noble, J., editor, *ECOOP*, volume 7313 of *Lecture Notes in Computer Science*, pages 104–131, Berlin. Springer

behave exactly the same as their counterparts implemented in R.

3.4.2 *Software Engineering Considerations*

To understand RGP's architecture from a software engineering perspective, it is important to understand some of the design goals underlying the R system that RGP is based on. Morandat et al. [2012] provide an in-depth treatment of R's architecture and semantics, including an evaluation of some of R's architectural features that is based on data gathered from a large body of real-world R source code.[36] R consists of four separate entities: 1) A *dynamically typed, lazy-evaluated, data-parallel, functional programming language* with extensions for *object-oriented* programming, 2) an interpreter implementing this language, 3) an environment for statistical computing and graphics, providing a wide variety of statistical and graphical techniques including linear and nonlinear modeling, statistical tests, time series analysis, classification, clustering, and 4) a collection of more than 5800[37] actively maintained extension packages implementing a vast number of techniques for data analysis, modeling, and visualization. At the time of writing, the only R interpreter widely adopted seems to be the official R interpreter available at `http://www.r-project.org`, although alternative implementations exist and are actively developed. This is why the terms "R language", "R interpreter", and "R ecosystem" are often used interchangeably.

[36] Morandat, F., Hill, B., Osvald, L., and Vitek, J. (2012). Evaluating the design of the R language - Objects and functions for data analysis. In Noble, J., editor, *ECOOP*, volume 7313 of *Lecture Notes in Computer Science*, pages 104–131, Berlin. Springer

[37] as of January 2014

In the following, some specific architectural features of RGP, that are sometimes inherited from R, are highlighted. It is shown how these features contribute to the high-level design goals introduced in the previous section, as well as how these features simplify the use and development of RGP in general.

Interactive Environment As shown in Section 3.2, data analysis and modeling is typically an exploratory and iterative process. A process of this kind greatly benefits from tools that support a highly-interactive workflow. The R system supports such a workflow through various architectural features. First, the R interpreter provides a read-eval-print loop (REPL) for the interactive evaluation of R expressions. This REPL has already been demonstrated extensively through the tutorial examples of Section 3.3. Second, the R language design is optimized for interactive use, sometimes even at the cost of robustness. Being a dynamically typed language, no type annotations are necessary. Also, variables are created at first assignment and need not be declared. Identifiers are resolved dynamically at runtime, meaning that programs can be changed while running. Third, the R environment provides additional features for the inspection of running programs, e.g., the built-in `debug` and `browser` functions. Combined, these features enable an efficient and highly-interactive workflow both for RGP users and RGP developers. In summary, RGP's interactive environment contributes to the

high-level design goal of approachability and improves overall user efficiency.

Functional Programming The functional programming paradigm appears to be better suited to the implementation of GP systems, or of CI software in general, than the more prevalent paradigm of object-oriented programming. GP systems, as well as most CI systems, are naturally specified as compositions of functions, as evident in the specifications of Chapter 2. Using a functional *data flow architecture* instead of an object-oriented *pipes and filters architecture* to implement these specifications seems both simpler and more productive. Furthermore, algorithm components of CI systems are naturally parametrized with functions. Examples include GP fitness functions, the GP operators of mutation, recombination and selection, and also activation functions of artificial neural networks, as well as membership functions of fuzzy sets, and kernels of support vector machines. Because first-class functions are an essential abstraction for the implementation of CI systems, functional programming can lead to a significant productivity gain. The reduction in mutable state and hidden side-effects inherent in the functional programming paradigm helps with RGP's high-level design objectives of extensibility and modularity. Additionally, a reduction or total lack of function side-effects greatly simplifies parallelization, contributing to the high-level design objective of efficiency. Hughes [1989] classical essay on why functional programming matters is still valid today and contains additional arguments for the usefulness of the functional paradigm. The implementation of RGP's web-based user interface also gains simplicity and clarity from the use of functional reactive programming, a specialization of functional programming geared to the development of interactive systems.

Domain-Specific Languages RGP extends R's characteristic use of *internal domain-specific languages* (DSLs) to enhance user experience and efficiency through short and familiar notation. An internal DSL is a DSL embedded into a host programming language by being built from host language constructs. This has the benefit that a multiple DSLs can be used to code different parts of the same program, each optimized to their respective domain. By contrast, an *external* DSL is built much like any other stand-alone programming language, using tools like parser generators or compiler compilers. R is a language in the tradition of Lisp (and Scheme) and thus very well suited to host internal DSLs. For instance, R enables program access to the parser via the `parse` function, to the interpreter via the `eval` function, and to the execution environment via the `environment` function. Furthermore, R source code expressions can be easily processed by R code itself, a property often referred to as "computing on the language". RGP embraces internal DSLs at various locations. The use of the familiar formula notation for

specifying symbolic regression models has already been mentioned previously. RGP provides an additional internal DSL for specifying types of R expression, which was shortly mentioned in Section 3.3.4. As the concept of an internal DSL is common and natural in the R ecosystem, RGP uses small DSLs to simplify many tasks. For example, evolution stop conditions can be combined through logical expressions using a basic internal DSL. RGP's web-based user interface is implemented in *Shiny*, an internal DSL for functional reactive programming.[38] The practice of combining multiple internal DSLs in a single program is also known as *language-oriented programming*.[39] In RGP, language-oriented programming is employed to further the high-level design objectives of approachability and extensibility.

[38] RStudio Inc. (2014). *shiny: Web Application Framework for R*. R package version 0.10.1, http://CRAN.R-project.org/package=shiny (retrieved 20.02.2015)

[39] Ward, M. P. (1995). Language oriented programming. *Software – Concepts and Tools*, 15(4):147–161

3.5 RGP Features

With 193 public, i.e., end user accessible, functions[40], RGP is large enough to be a bit daunting to first time users. Detailed documentation for each RGP function is available through R's online documentation system by issuing the command `help(package="rgp")` in a running R session. Additionally, a set of introductory tutorials is available via the command `vignette("rgp_introduction")`. This section serves as a link between introductory tutorial examples and reference documentation by providing structure and background information to RGP's functionality and by connecting the abstract specifications of Chapter 2 to concrete RGP implementations.

[40] as of RGP release 0.4-0

Figure 3.9 shows the top-level organization of the RGP package. Each source code module is represented by a box. Box sizes correspond to the number of public functions defined in a source code module.

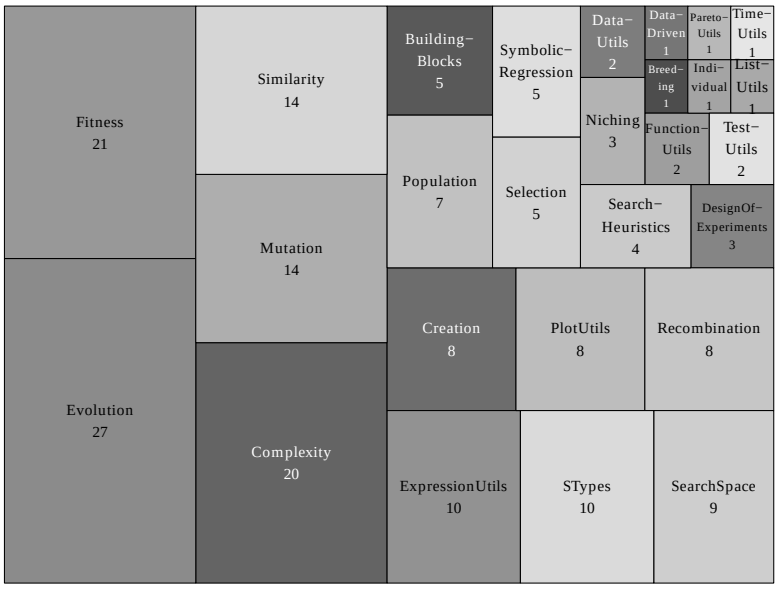


Figure 3.9: RGP Source Code Treemap. This figure shows the top-level organization of the RGP package. Each source code module is represented by a box. Box sizes correspond to the number of public functions defined in a source code module.

3.5.1 GP Individual Representation

As already mentioned, RGP employs R expressions to represent GP individuals. Because R expressions are represented internally as trees, RGP basically is a tree-based GP system. R supports the direct manipulation of expression trees through the same syntax used for manipulating nested lists, making the implementation of GP operators in R simple, concise, and easy-to-understand for users familiar with R. Another benefit of this representation is its compatibility with the R ecosystem. RGP can directly use all available R functions, either built-in or available on CRAN, as members of GP function or constant sets. Furthermore, solutions generated by RGP are standard R functions and can therefore be processed by many R functions. For example, symbolic regression models created by RGP may be directly plotted not only by R's built-in generic `plot` function, but also by many of the specialized plotting and visualization functions available on CRAN.

A drawback of using R expressions as RGP's standard individual representation is the comparatively low performance of the R interpreter, at least at the time of this writing. RGP uses multiple techniques to lessen the performance impact this incurs. First, R's vector processing capabilities are used as much as possible. For example, in symbolic regression and classification, all fitness cases, i.e., all data rows, are evaluated in a single invocation of the R interpreter. Second, performance-optimized versions of commonly used GP operators exist. These are implemented in C and bypass the R interpreter. Third, RGP supports the addition of alternative representations to enable the implementation of performance-optimized GP *kernels*.[41]

[41] A GP kernel is a matching set of representation, operators, and search strategy.

Alternative representations can be used in RGP by providing initialization, variation, and evaluation operators for these representations. One of the examples included in the RGP package shows how to implement a linear GP representation. The recommended strategy for implementing alternative representations for RGP, and the strategy used in this example, is to first implement a pair of functions to convert RGP individuals to and from the alternative representation. After wrapping existing operators with these conversion functions, RGP is already operational. Operators specialized to the alternative representation may now be added piece by piece, simplifying implementation and testing considerably.

3.5.2 RGP Type System

The RGP type system has been developed with the following design goals in mind:

- *Approachability:* Type terms (the language of types) should be easily formulated by non-experts.
- *Expressiveness:* The search space defined by the typed building blocks should be as large as necessary, but not larger, while still allowing the efficient implementation of search operators.

- *Efficiency:* Creating and maintaining well-typed solutions should be efficient from a computational standpoint, i.e., no algorithms with exponential runtime should be used.
- *Simplicity:* The type system should be easy to understand and implement.

As already mentioned, the *simply typed lambda calculus* forms the theoretical basis for RGP's type system.[42] The type system supports base or atomic types such as `st("logical")` or `st("numeric")`, and function types such as `list(st("numeric"), st("numeric")) %->% st("numeric")`, but no generic types. Syntactically, the set of RGP type terms T_R are defined recursively as follows:

[42] Barendregt, H., Abramsky, S., Gabbay, D. M., Maibaum, T. S. E., and Barendregt, H. P. (1992). Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309, New York. Oxford University Press

$$\texttt{st(}\textit{type name}\texttt{)} \in T_R \qquad \text{(base types)}$$

$$\forall t_1, \ldots, t_n \in T_R : \texttt{list(}t_1, \ldots, t_{n-1}\texttt{) \%->\%} \; t_n \in T_R \quad \text{(function types)}$$

Semantically, this type system can express the type of all pure non-generic R functions. It is not limited to RGP individuals, but can be applied to define and calculate types of a large subset of R expressions. Also, it is conceptually simple and relatively easy to implement efficiently.

As a consequence of R's support for anonymous functions and the RGP type system's support for function types, *function defining subtrees*, i.e., anonymous functions or lambda abstractions, can be evolved through RGP. This offers a flexible alternative to the concept of *automatically defined functions*.[43] It also allows the integration of common higher order functions like folds, mappings, and convolutions into the set of GP building blocks.

[43] Koza, J. R. (1994). *Genetic programming II: Automatic discovery of reusable programs*. MIT Press, Cambridge, MA

Internally, RGP infers the types of compound expressions such as function applications and function definitions from the types of atomic expressions. This inference is done when type information is needed by a GP operator. The types of building blocks are defined by the user via the `%::%` operator and are stored in the package-internal global variable `rgpSTypeEnvironment`. RGP's `sType` function can be used to calculate the type of a given R expression. Type information is cached for efficiency. If no type information is supplied for the building block set, the type system in deactivated and untyped variants of all GP operators are used, which are more efficient. This is useful for typical untyped GP tasks such as symbolic regression based on numerical data. See Section 3.3.4 for a tutorial example on strongly typed GP with RGP.

3.5.3 *Initialization, Variation and Selection Operators*

RGP provides implementations of the initialization operator defined in Section 2.6.1. Additionally, an optimized variant written in C is available, as well as a type safe variant. This variant only creates R expressions that are well typed, given that the type of each member of the constant, variable, and function set has been defined by the user via the `%::%` operator, as described in the previous sub-

section. The RGP online documentation on the functions `randfunc` and `randfuncTyped` provides additional usage details.

RGP also includes implementations of the mutation and crossover operators defined in Sections 2.6.2 and 2.6.3. As with initialization, performance optimized variants and type safe variants are available. See the documentation on the functions `mutateFunc` and `crossover` for details.

The single- and multi-objective selection operators described in Section 2.6.4 are available. Multi-objective selection is provided by the `emoa` package.[44]. Additionally, single- and multi-objective tournament selection operators are available. For details, see the online documentation on the functions `nds_cd_selection` and `makeTournamentSelection`.

[44] see `http://cran.r-project.org/web/packages/emoa/` for details

Each GP operator provided by RGP provides support for *breeding*, i.e., the repeated application to optimize a quality criterion of the result. See RGP's online documentation on the respective GP operator for details.

3.5.4 *Quality Measures and Fitness Functions*

All fitness functions and quality measures defined in Section 2.5.3 are efficiently implemented by RGP and described in the online documentation. This includes the error measures `mae`, `sse`, `rmse`, `smse`, and `rsquared`. The visitation length measure is implemented alongside other measures of solution compactness, see the online documentation on the function `funcSize` for details. Additionally, measures to approximate the GP search space size are provided. These measures calculate the number of structurally different expressions or expression shapes of a given depth or size that can be build from a fixed function- and input-variable set. Here, the term "expression shape" means the shape of an expression tree, not taking any node labels into account. See the online documentation on `exprShapesOfDepth` for details.

For convenience, RGP also includes factory functions for common fitness functions. For example, `makeNaryFunctionFitnessFunction` creates a fitness function that calculates an error measure with respect to an arbitrary n-ary reference function based on sample points generated by a given *design function*, which defaults to a function generating a regular grid design matrix. Another example is the fitness function factory `makeBooleanFitnessFunction` that, given a boolean target function, produces a fitness function that returns the number of different binary digits in the output of a given boolean function and a given target function.

3.5.5 *Search Heuristics*

RGP implements the search heuristics described in Section 2.9. Additional search heuristics can be added by the user without changing any RGP code. RGP's top-level functions for performing GP runs, i.e., `geneticProgramming`, `typedGeneticProgramming`,

and `symbolicRegression`, have a `searchHeuristic` argument. The search heuristics included in RGP are configured through factory functions. See the online documentation on the functions `makeTinyGpSearchHeuristic`, `makeCommaEvolutionStrategySearchHeuristic`, and `makeAgeFitnessComplexityParetoGpSearchHeuristic` for details. This level of flexibility regarding the search heuristic is unique among current GP systems.

3.5.6 RGP Runtime Environment

In this context, the term *runtime environment* designates functionality for supporting efficient GP run execution. This includes an optimized evaluator for a subset of R, mainly used to improve the execution time of symbolic regression runs. Details on the benefits and limitations of this optimized evaluator can be found in the RGP documentation.

Also part of RGP's runtime environment are tools for monitoring GP runs in progress and setting termination criteria. The latter is accomplished via a small internal DSL for defining stop conditions. In RGP, a stop condition is a predicate, i.e., a function returning a single boolean value. Included are stop conditions based on generations, fitness function evaluations, best fitness, and wall-clock time. See the online documentation on the RGP function `makeEvaluationsStopCondition` for details. Trough the internal DSL, stop conditions may be combined trough boolean operators. For example, the termination criterion defined as `makeTimeStopCondition(60) || makeFitnessStopCondition(0)` finishes a GP run after 60 seconds or when a solution of fitness 0 is found, whatever event occurs first.

Runs can be monitored by providing a monitor callback function to any of RGP's top-level functions for performing GP runs. See the online documentation on the parameter `progressMonitor` for details on this mechanism. The default monitor callback prints basic progress information to the system console in regular intervals.

3.5.7 Result Analysis

Analysis of GP runs in RGP is accomplished in large part through tools provided by R. This includes model validation tasks, such as assessing the interpolation and extrapolation quality of a model or set of models. Result analysis tasks not readily supported by R's base functionality are supported through RGP functions.

Model tables, listing all models resulting from a GP run, together with relevant meta-data such as validation fitness and complexity values, are part of the GP result data frame returned by RGP's top-level functions `geneticProgramming`, `typedGeneticProgramming`, and `symbolicRegression`. Pareto plots, showing the Pareto front of the set of result models of a GP run, are created via the RGP function `plotParetoFront`. Variable presence maps, showing the distribution of input variables used in the set of result models of a GP

run, are created via the RGP function `populationVariablePresenceMap`. Basic algebraic simplification of models created by RGP is provided by the Rrules package. Finally, RGP adds a method to R's generic function `summary` for summarizing the results of a GP run. Based on these functions, comprehensive result report documents can be compiled with help of R's knitr package for dynamic report generation.[45]

[45] Xie, Y. (2013). *Dynamic Documents with R and knitr*. Chapman and Hall, Boca Raton, FL

3.6 Symbolic Regression User Interface

This section contains a detailed description of RGP UI, RGP's web-based user interface for symbolic regression. The description is based on RGP UI release 0.1-1, which is current at the time of writing. RGP UI is meant to enable users not familiar with command-line user interfaces to successfully prepare, perform, and analyse symbolic regression runs with RGP. In its current state, only a careful selection of basic RGP features is accessible through the web-based user interface. Particularly, RGP's type system is not supported at this point in time, limiting RGP UI to numerical data. Within these limits though, RGP UI has already been successfully applied to real-world applications.

3.6.1 Installation and Start

Just as RGP itself, RGP UI can be automatically installed via CRAN. To do so, issue the following command in a running R session:

```
install.packages("rgpui")
```

If installation was successful, RGP UI can be started by issuing the following commands. Note that the second command will not return, which is due to the single-threaded nature of R's front-end user interface:

```
library("rgpui")
symbolicRegressionUi()
```

A web browser should open on the URL `http://localhost:1447` and display the web-based user interface. In case this does happen due to lack of system support, open a web browser and point it to the mentioned URL by hand.

3.6.2 User Interface Structure

The RGP UI user interface supports symbolic regression through a subset of the workflow described in Section 3.2 and shown in Figure 3.2. It is structured into a two-level hierarchy of panels and views. Each panel corresponds to a step of the RGP process for symbolic regression. Controls for data entry are shown on the left side, while view for data visualization are shown on the right side. Each panel contains short documentation marked with an info symbol. Figure 3.10 shows the hierarchical structure of RGP UI in

form of a tree. The following paragraphs describe each panel in detail.

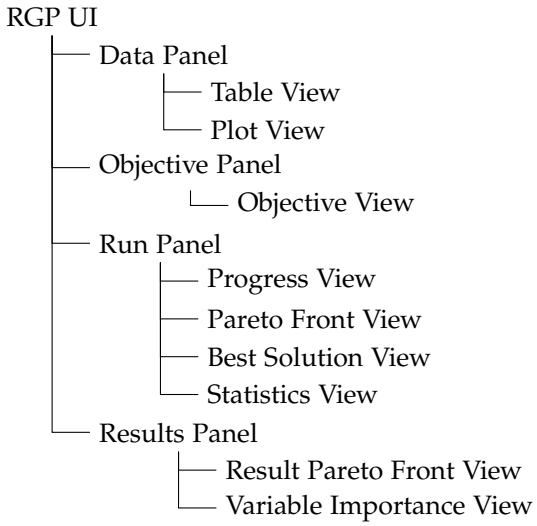


Figure 3.10: Structure of the RGP UI web-based user interface. Each panel corresponds to a step in the RGP process for symbolic regression. Typically, the panels shown in this tree are visited from top to bottom when performing symbolic regression. Some panels offer multiple data visualizations, represented in this tree as multiple views.

Data Panel As RGP UI is currently limited to symbolic regression on existing data, it is assumed that the steps of regression problem definition and data acquisition and data preprocessing have already been completed with exception of data partitioning, which should be done in RGP UI. The data should be available in CSV format. Figure 3.11 shows the data panel. The first step of symbolic regression in RGP UI is to upload data via the "select file" button. Once the upload completes, a data table and plot become available through the table and plot views. Specifics of the CSV format can be configured on the right, the views will update instantly to give direct feedback on the correctness of the data import settings. The next step is to partition the data into training and validation sets through the data partitioning controls to the right. Training data can be drawn at random (with configurable random seed for reproducible results), or be the first, middle, or last data rows. The latter options are valuable when testing the extrapolation quality of a model.

Objective Panel The objective panel, as shown in Figure 3.12, allows the definition of dependent and independent variables, the error measure, and the building block set. Additionally, an option for enabling or disabling the complexity criterion of the multi-objective selection operator is provided. The default settings are chosen based on the data imported in the previous step. The view to the right of the panel shows a plot of the dependent variable against the row number. As in the plot view of the data panel, points in the training set are shown as solid dots, while points in the validation set are shown as hollow dots.

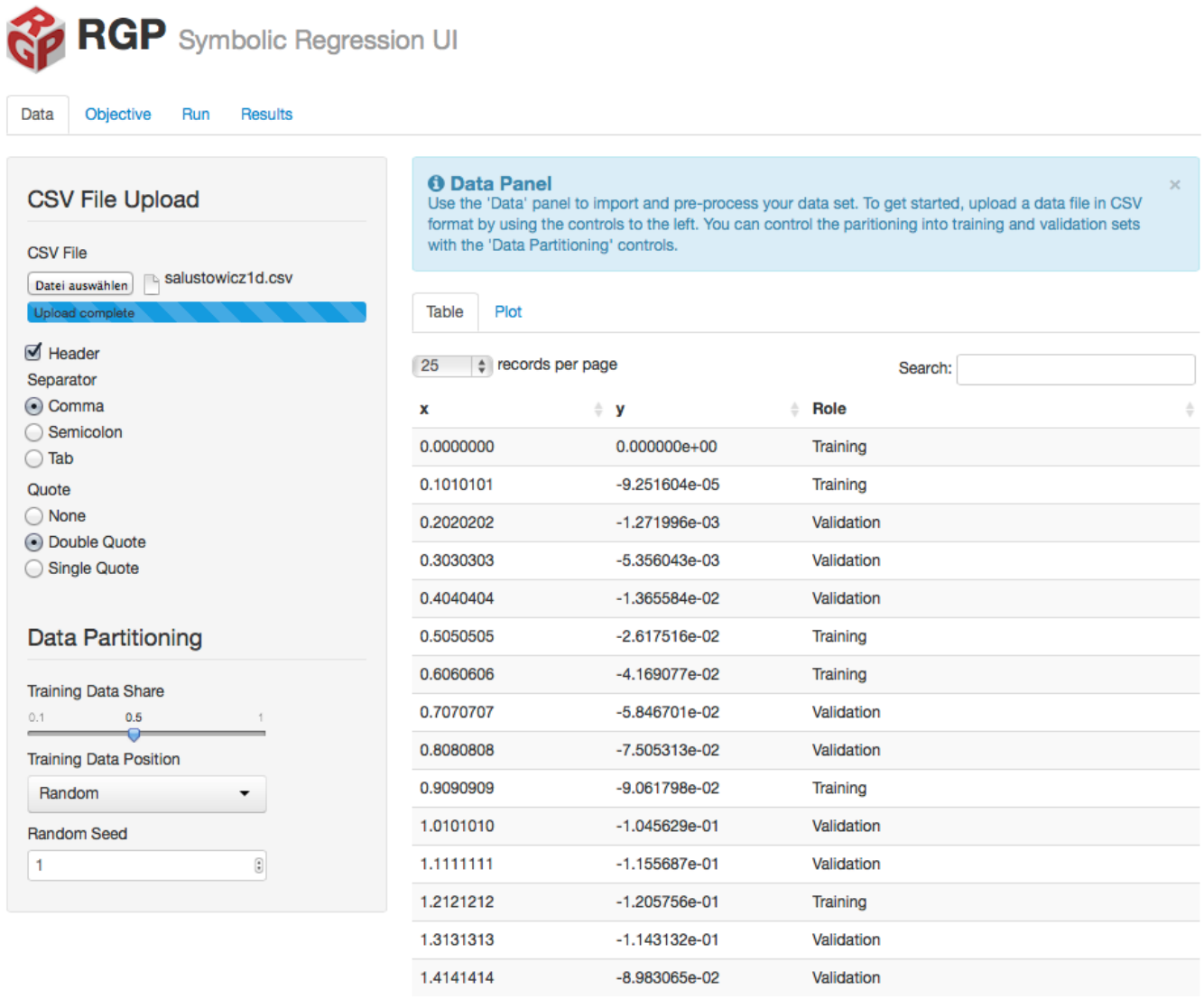


Figure 3.11: RGP UI Data Panel. In this panel, data can be uploaded and imported. Also, data partitioning into training and validation sets is performed here. Table and plot views provide live visual feedback to assure the correctness of the data import settings.

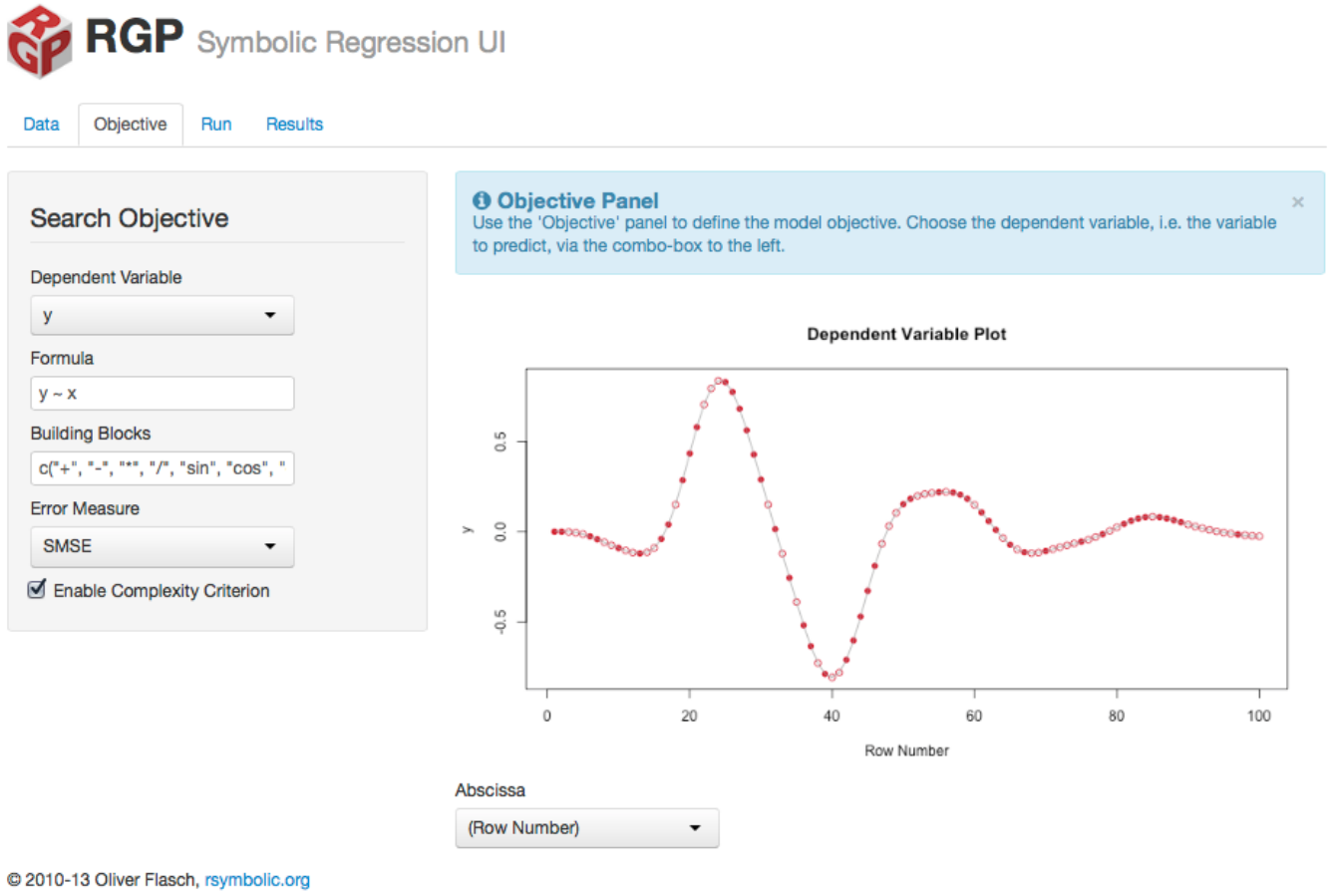


Figure 3.12: RGP UI Objective Panel. This panel allows the definition of dependent and independent variables, the error measure, and the building block set. Additionally, an option for enabling or disabling the complexity criterion of the multi-objective selection operator is provided. As in the plot view of the data panel, points in the training set are shown as solid dots, while points in the validation set are shown as hollow dots.

Run Panel The run panel, shown in Figure 3.13, provides controls for starting, pausing, and resetting RGP symbolic regression runs. A run has to be paused to enable the result panel described in the next paragraph. Resetting a run re-initializes the solution population. The left side of the run panel also controls the most important parameters of the GMOGP search heuristic used in RGP UI. These include population size (μ), number of children per generation (λ), number of new individuals per generation (ν), probabilities for the crossover and specific mutation operators, an option to enable or disable age layering, the probability of performing parent selection at each generation, and the random seed to use for the run. The right side of the run panel contains four views: The progress view contains plots of multiple GP run performance metrics against time measured as number of generations, including fittest individual fitness, fittest individual complexity, fittest individual age, and dominated hypervolume. The Pareto front view shows a plot of the current population, where the x-axis denotes fitness (prediction error) and the y-axis denotes complexity (visitation length). Individual age is shown through color, where solutions turn from green to blue to black with increasing age. The best solution view shows the best solution (blue) and the data (red) overlayed in the same plot, as well as the best solution's symbolic expression, and the generation number, fitness evaluation number, and time elapsed when it was found. Finally, the statistics view shows information about the current run, including current generation, fitness evaluation number, time elapsed, and the current number of fitness evaluations per second.

Results Panel The results panel, shown in Figure 3.14, shows a report of the results of the current symbolic regression run. The run has to be paused via the run panel for this report to become available or to be updated. The results panel does not contain any input controls. Two result report views are available: The result Pareto front view contains a sortable table of the result expressions part of the current Pareto front. This table contains the columns *Formula* (result symbolic expression), *Error (Training)* (prediction error on training data), *Error (Test)* (prediction error on validation data), R^2 (R^2 on validation data), *Complexity* (visitation length), and a small overlayed plot of the solution versus the data, as known from the run panel's best solution view. The variable importance view shows a bar plot of the number of occurrences of each independent variable in the result population. The part of each bar denoting occurrences in Pareto front individuals is colored red, the part denoting occurrences in other individuals is colored blue. With higher-dimensional data, this plot is a useful tool for quantifying the influence of each independent variable on the dependent variable.

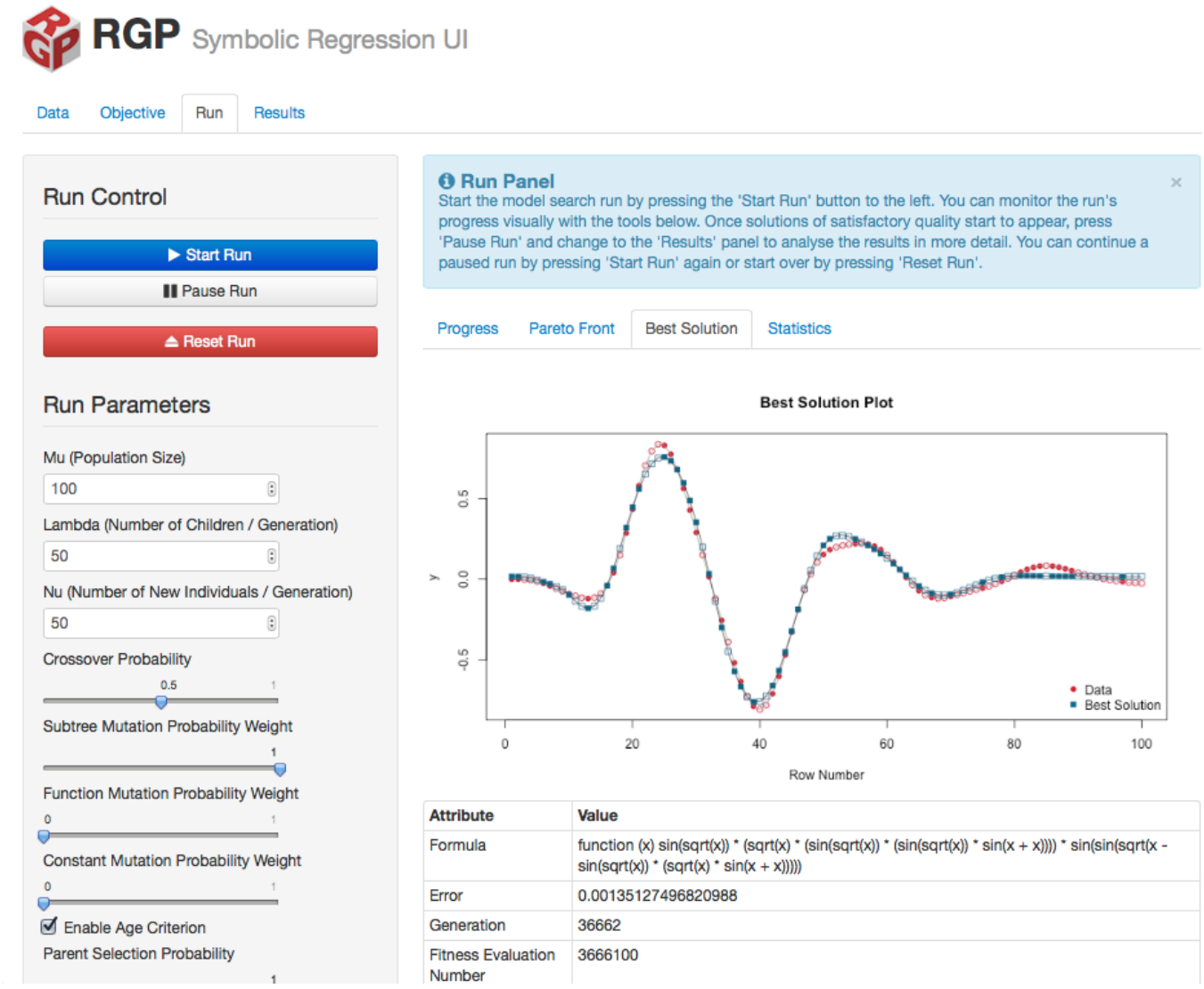


Figure 3.13: RGP UI Run Panel. This panel provides controls for starting, pausing, and resetting RGP symbolic regression runs. The left side controls search heuristic parameters such as population size and number of children per generation. The right side contains the progress view, the Pareto front view, the best solution view, and the statistics view.

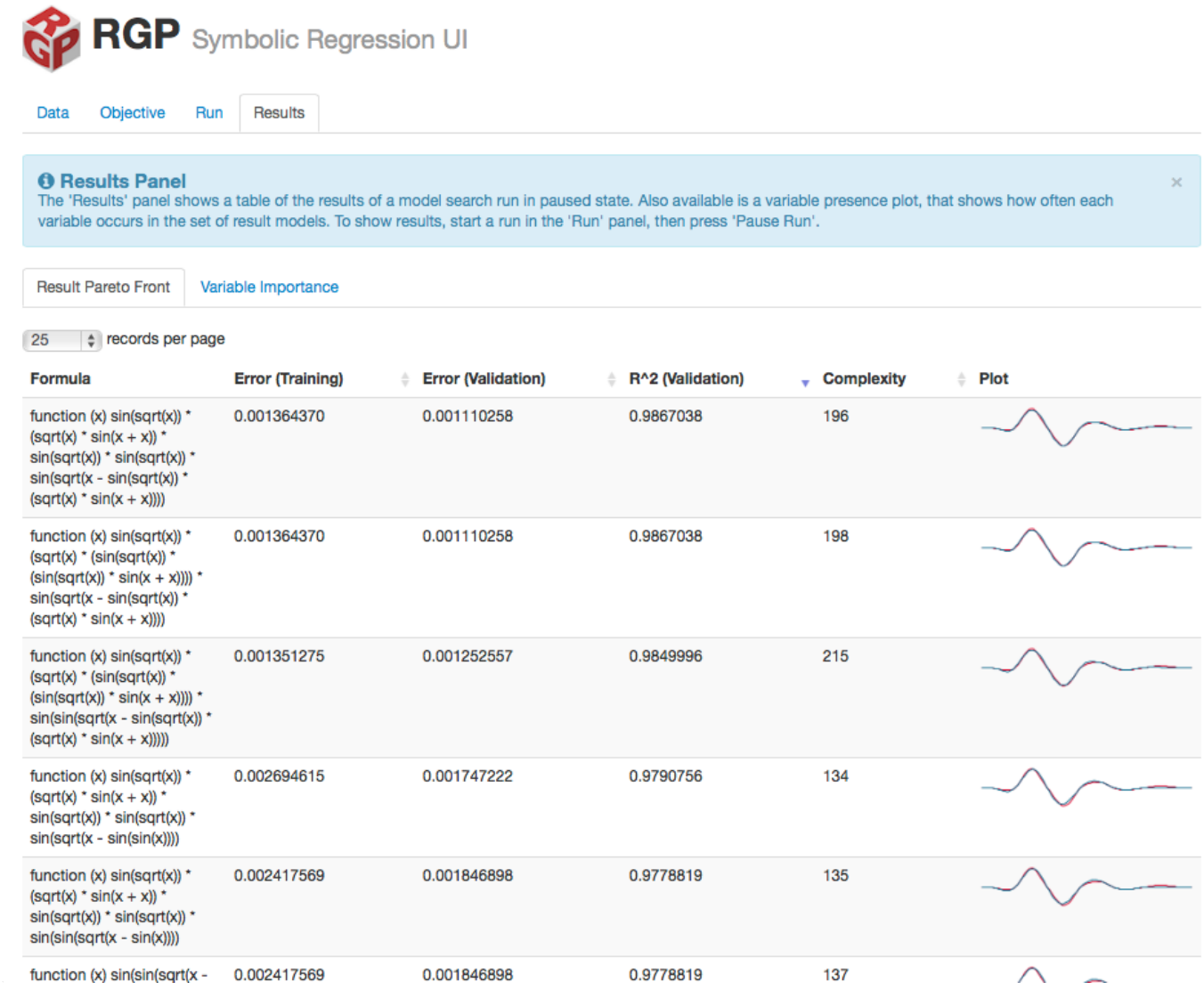


Figure 3.14: RGP UI Results Panel. This panel shows a report of the results of the current symbolic regression run. The run has to be paused via the run panel for the result panel to become active. The result report is divided into the result Pareto front view and the variable importance view.

3.7 Conclusions

This chapter introduced the design and implementation of RGP and the web-based user interface RGP UI. The high level of integration into the R environment allows RGP to focus on the core functionality of a modern GP system, while profiting from the extensive tools for preprocessing and result analysis available in the R ecosystem.

4
Sequential Parameter Optimization

Sequential parameter optimization (SPO) is a framework for understanding and improving the behavior of search and optimization algorithms by experimental methods. Classical as well as modern statistical methods are employed to optimize algorithm parameters and to improve performance and robustness. SPO sequentially performs a preset number of experiments, i.e., algorithm runs, and uses the algorithm's response to its parameter settings to build and refine one or more meta-models of the parameter search space.

Goal of this chapter is to introduce necessary tools for the systematic empirical analysis of GP system components and their effect on GP system performance. Here, a GP system component is a part of GP system that can be changed or swapped meaningfully. Examples include GP search operators, individual representations, and GP search heuristic as defined in Chapter 2.

Performing a thorough and statistically well-founded experimental analysis provides valuable insight into the behavior of GP system components. In current GP research, much repeated work in experimental planning, setup, and result analysis is required when proposing improvements for GP system components such as selection and variation operators, individual representations, or search heuristics. To measure the performance benefit of an improved GP system component, a set of test functions has to be implemented, GP system parameters, both of the system under study as well as of comparison systems, have to be chosen, experiments have to be designed and code for the statistical result analysis has to be written. As obtaining results of statistical significance often requires many independent runs, infrastructure for automated distributed execution is often necessary to render the implementation of an experiment plan practical.[1]

Time spent re-implementing necessary GP research infrastructure is lost for working on core issues. As each group working in GP research has to provide not only their own GP system components, but also complex supporting infrastructure often tightly coupled to local conditions, reproducing results by other groups is often more difficult than necessary. The open-source tools introduced in this chapter may serve as a first step towards a framework of reusable building block for reproducible research in GP.

[1] In the field of continuous global optimization, the COCO framework for comparing continuous optimizers provides a well-established solution to this problem. While this framework could be adapted for GP by recasting the provided test problems as symbolic regression problems, the resulting test problems would not be representative for typical real-world GP tasks.

Hansen, N., Finck, S., and Ros, R. (2011). Coco - Comparing continuous optimizers: The documentation. Technical Report RT-0409, INRIA, France

The framework for reproducible research in GP consists of two parts:

- software for adapting SPOT to RGP, enabling parameter analysis and tuning for arbitrary RGP configurations
- a set of scalable randomized test functions for accessing the performance of GP system components and their parameter settings in a controlled and reproducible manner

IDEAS and concepts regarding SPO are borrowed from the empirical approach to research in evolutionary computation introduced by Bartz-Beielstein et al. [2005].[2] A preliminary version of the approach to scalable test problems introduced in this chapter has been previously published by Flasch and Bartz-Beielstein [2012].[3]

[2] Bartz-Beielstein, T., Lasarczyk, C., and Preuss, M. (2005). Sequential parameter optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 773–780, Piscataway, NJ. IEEE Press

[3] Flasch, O. and Bartz-Beielstein, T. (2012). A framework for the empirical analysis of genetic programming system performance. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice X*, Genetic and Evolutionary Computation Series. Springer, New York

4.1 SPO *Process and Optimization Objectives*

After the last section introduced SPO and the problems it solves in the context of this work, this section describes the SPO process and SPO optimization objectives in more detail.

4.1.1 SPO *Process*

The general process of SPO consists of the following four steps:

1. Formulate a scientific question to be analyzed by statistical means.
2. Divide the complex question into simple statistical hypotheses that can be conquered by statistical testing.[4]
3. For each of these hypotheses:
   (a) Select a model/predictor (e.g. a linear model or a regression tree) to describe a functional relationship between parameters and response.
   (b) Select an experimental design.
   (c) Generate experimental data by running experiments.
   (d) Refine the model/predictor until the hypothesis can be accepted/rejected by statistical testing.
4. Analyze the experiment results in order to draw conclusions from the hypotheses to assess their scientific meaning. At this step, the concept of severe statistical tests, first introduced by Mayo [1996], is of high relevance. Whether an hypothesis H passes an appropriate statistical test is a *severe test of* H only to the extend that it is very improbable (e.g. five in hundred or one in hundred) that the test were passed if H were false.

   Figure 4.1 shows a flow diagram of the SPO process.

[4] See also Mayo [1996] for a detailed discussion of the theoretical background.

4.1.2 SPO *Objectives*

The objective of SPO in the context of this work is to find near-optimal parameter settings for a given stochastic search algorithm

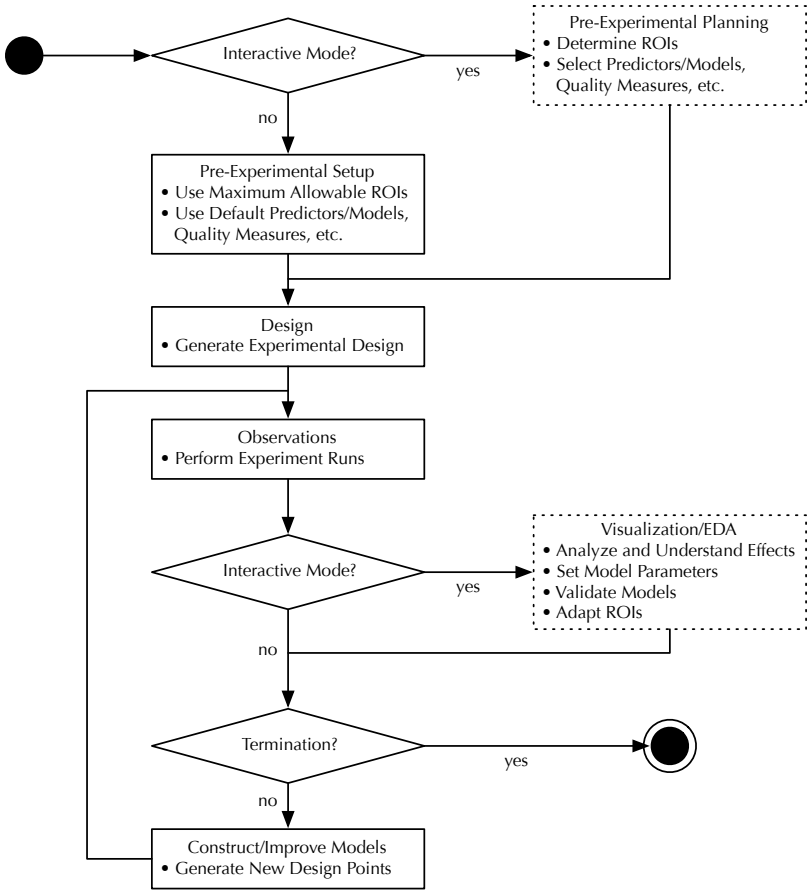


Figure 4.1: The SPO process, presented as a flow chart. This figure originally appeared in Bartz-Beielstein [2009].

A. Here, optimality in can be measured by several means, including algorithm performance on test data, or algorithm robustness, i.e., variance in algorithm performance on different input data.

SPO treats the algorithm A as a black box. For each run of A, a vector of input variables $x \in X$ is passed to the algorithm, which produces some output $y \in Y$. SPO then tries to determine a functional relationship F between X and Y, i.e., fits a model F to the parameter-response pairs (x, y). During this process, the effect of pseudo-randomness has to be taken into account if

i) the objective function f of the search algorithm A is stochastically disturbed,
ii) the search algorithm A itself includes stochastic elements, e.g. random mutation in evolutionary search strategies.

The elements of the set X can be further classified in the following manner:

1. Parameters needed by the algorithm A belong to the *algorithm design*, whereas
2. variables needed to specify the objective function f[5] belong to the *problem design*.

The set X is then divided into the algorithm design X_a and the problem design X_p.

[5] or, in other words, the optimization problem f

4.2 *Sequential Parameter Optimization Toolbox*

The *sequential parameter optimization toolbox* (SPOT) provides implementations of the SPO process both in R and in MATLAB. In this work, the R implementation has been used exclusively.

SPOT is not a meta-algorithm exclusively, but also a system for performing EDA on the algorithm designs tried in an SPO run. In this regard, SPOT does not only provide a solution for the algorithm tuning problem, but also insight into the algorithm response to different parameter settings, which will be demonstrated in the next chapter. The system provides tools for performing the following tasks corresponding to individual steps in SPO process:

1. *Initialize* creates an initial experiment design, stores it in memory and optionally writes it to a design file. Usually, this is the first step of a SPO run. This step uses information on the type and region of interest for each algorithm parameter. It is related to the data acquisition step of Figure 3.2 on page 63.

2. *Run* performs algorithm runs with parameter settings from the initial experiment design created in the initialize step. Results are stored in memory and optionally written to a result file. This is usually the second step of a SPO run. It uses information on both the algorithm and problem design. A model/predictor F is fitted based on the results. SPOT includes several generic models/predictors to use and is easily extensible with new models. It corresponds to the modeling step of Figure 3.2 on page 63.

3. *Sequential Step* creates a new experiment design based on information gained from the previous run step, stores it in memory and optionally writes it to a design file. Information from the model/predictor F is used to generate the new design.

4. *Report* generates a statistical analysis based on the results obtained in the previous run and sequential steps. The report contains basic regression analysis and visualizations such as scatter plots, histograms, residual plots and more. Report generation is based on a LaTeX template and is easily extensible. This step corresponds to the analyze results step of Figure 3.2 on page 63.

In *Automatic Mode*, the initialize step, followed by a predefined number of run/sequential steps and finalized by the report step, are performed in an automatic manner, suitable for unattended runs. This mode is especially useful in cases where individual algorithm runs are computationally expensive, as it is the case in GP algorithm tuning, and sufficient parallel computing resources or compute time resources are available.

THE SPO process implemented by SPOT is best illustrated by means of examples. Bartz-Beielstein and Zaefferer [2012] provide two step-by-step tutorials for SPOT, the first focusing on parameter

tuning for the simulated annealing algorithm built into R's `optim` function, the second on tuning a simple ES.[6] Both examples are recommended as they illustrate SPOT in much more detail than possible in the space constraints of this chapter. They can serve as a bridge to understanding the more complex SPO use cases described in the next chapter on analysing and tuning GP parameters.

[6] Bartz-Beielstein, T. and Zaefferer, M. (2012). A gentle introduction to sequential parameter optimization. Cologne Open Science, Schriftenreihe CI-plus TR 01/2012, Bibliothek der Fachhochschule Köln, Betzdorfer Str. 2, 50679 Köln, Germany

4.3 *Scalable Random Test Problems*

Test problems of controllable difficulty, i.e., *scalable test functions*, are a flexible tool for assessing the relative performance benefits of different GP system components under varying conditions. Unfortunately, defining finely scalable test functions for GP holds many challenges. Conventional measures of problem difficulty, such as information criteria[7], that are good predictors for the performance of fixed-structure modeling approaches, such as statistical regression techniques, often fail to predict the performance of GP runs.[8] At least in the important application of symbolic regression, test functions of comparatively simple structure can prove extremely difficult for state-of-the-art GP systems.[9]

[7] Kieseppä, I. A. (1997). Akaike information criterion, curve-fitting and the philosophical problem of simplicity. *British Journal for the Philosophy of Science*, 48(1):21–48

[8] See Section 2.5.3 for definitions of performance measures for GP.

[9] Korns, M. (2011). Accuracy in symbolic regression. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation Series, pages 129–151. Springer, New York

The following subsections describe the two kinds of scalable test problems used in the remainder of this work:

1. Scalable test functions with spurious variables and
2. scalable random test functions based on Kriging models.

Note that both kinds are useful on their own, but could also be combined, as they differ in their approach to problem difficulty scaling.

4.3.1 *Spurious Variable Test Functions*

This class of scalable test problems is based on the simple yet very effective idea first described by Kotanchek et al. [2006] of adding spurious variables to the set of fitness cases to conceal the true functional dependency between driving variables and output for arbitrary test functions.[10] To increase the difficulty of a given function, additional function arguments (spurious variables) are added that do not influence the function value at all. As this fact is not known to the GP system, and spurious variables might be correlated with the function value by chance, problem difficulty can be increased stepwise by increasing the number of spurious variables. Of course, spurious variables can also be added to input data directly, without resorting to a known test function, enabling scalable test problems based on real-world data.

[10] Kotanchek, M., Smits, G., and Vladislavleva, E. (2006). Pursuing the Pareto paradigm: Tournaments, algorithm variations & ordinal optimization. In Riolo, R. L. et al., editors, *Genetic Programming Theory and Practice IV*, Genetic and Evolutionary Computation Series, pages 167–186. Springer, New York

In the experiments conducted for this study, the spurious variable test functions described below were used. For each test function, one to ten spurious variables were added to gradually increase problem difficulty. Fitness cases were created by uniform random sampling in the indicated training and validation intervals. Training and validation set sizes are indicated below.

P1 *(Simple Sine)* Discovering the input-output relation of the sine function should be trivial even for a very simple GP system configured for symbolic regression, if the sin function is included in the GP function set and no spurious variables are introduced.

$$f_{\mathrm{P1}}(x_1) := \sin(\pi x_1)$$

The factor π has been introduced to test the GP system's capability of fitting constants. The training interval of the Simple Sine test function is fixed to $x_{\text{train}} := [-\pi, \pi]$, the validation interval is fixed to $x_{\text{validation}} := [-\frac{3}{2}\pi, \frac{3}{2}\pi]$, and the number of fitness cases, i.e., the test function sample size, is fixed to $N_{\mathrm{P1}} := 32$, based on results of a pre-experimental study conducted to avoid floor or ceiling effects. The GP function set used with this test function is $\{+, -, *, /, \sin\}$.

P2 *(Newton Problem)* Compared to the Simple Sine test function, the Newton Problem poses a slightly harder test case, as the true expression has slightly larger genotypic size. It was included as a more practical example based on a well-known natural law.

$$f_{\mathrm{P2}}(x_1, x_2, x_3) := \frac{x_1 x_2}{x_3^2}$$

The training interval of this test function is $x_{\text{train}} := (0, 1]$, the validation interval $x_{\text{validation}} := (0, 2]$, and the number of fitness cases is fixed to $N_{\mathrm{P2}} := 64$, based on results of a pre-experimental study. The GP function set used with this test function is $\{+, -, *, /\}$.

P3 *(Sine Cosine)* This test function is a simplified variant of test problem *(P12)* given in Korns [2011]'s work on accuracy in symbolic regression.[11] It is considered difficult for GP because it contains constants as arguments to non-linear functions.

$$f_{\mathrm{P3}}(x_1, x_2) := 6 \sin(x_1 - 3) \cos(x_2 - 3)$$

The training interval of the Sine Cosine test function is $x_{\text{train}} := [-\pi, \pi]$, the validation interval $x_{\text{validation}} := [-\frac{3}{2}\pi, \frac{3}{2}\pi]$, and the number of fitness cases is fixed to $N_{\mathrm{P3}} := 128$, based on results of a pre-experimental study. The GP function set used is $\{+, -, *, /, \sin, \cos\}$.

[11] Korns, M. (2011). Accuracy in symbolic regression. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation Series, pages 129–151. Springer, New York

4.3.2 *Kriging-Based Test Functions*

The scalable test functions based on spurious variables described in the previous subsection are simple to understand and implement, but offer only a single variant per difficulty level. When used for GP system parameter tuning and analysis, this can become an issue, as analysis and tuning results might be overfitted to the concrete test function. Possible solutions include using a different test function for validation, or creating variants by adding random noise. Unfortunately, these solutions have issues. Typically, optimal GP parameter settings are not entirely problem independent. This

fact is easily demonstrated by adding the target function to the GP function set, which is a GP system parameter. Swapping the target function for parameter validation to a completely unrelated one is therefore bound to give results of limited meaningfulness. Adding random noise to create test function variants poses another set of problems, including how to choose the noise amplitude. It can be assumed that symbolic regressions problems from the same application domain have similar properties, which are diluted by simply adding random noise.

Kriging-based scalable random test functions proposed here offer an alternative solution that does not suffer from these problems. They also offer additional benefits, including a new approach to problem difficulty scaling complimenting the spurious variable approach. A drawback of their current implementation is that they can only be applied to symbolic regression problems based on numerical input data.

Kriging is an interpolation method that, under certain conditions, gives the best linear unbiased prediction of the intermediate values. It is widely used in surrogate modeling and has properties that make it equally useful as the basis of a scalable random test function generator. Forrester et al. [2008] (pp. 51–63) define the Kriging model building and prediction process.[12] See Section 6.2.6 for additional details.

[12] Forrester, A., Sobester, A., and Keane, A. (2008). *Engineering Design via Surrogate Modelling*. Wiley, Chichester, U.K., 1st edition

The Kriging-based scalable random test function generator introduced here creates arbitrary many variations of a given base function or regression input data set. The range of complexity, or difficulty, of these variations is controlled by the parameter pair $(\delta_{\min}, \delta_{\max})$, where $\delta_{\min} \in mathbbR^{>0} \leq \delta_{\max} \in \mathbb{R}^{>0}$. Problem variation can be disabled by setting $\delta_{\min} = \delta_{\max}$.

Kriging-based scalable random test functions are constructed through the following algorithm:

1. A number of samples of the base function or regression input data set is designated as *Kriging model training data*.
2. A Kriging model is fitted on this training data.
3. The Kriging model's correlation parameter vector θ is perturbed by scaling with a uniform random vector of factors from the interval $[\delta_{\min}, \delta_{\max}]$.
4. The perturbed Kriging model fit is returned as a test function. If needed, it is sampled to gain a new regression input data set that is a variation of the base input data set.

Figure 4.2 shows how this process is implemented in practice. A Factory function creates a Kriging test function generator based on Kriging model training data and a difficulty range $[\delta_{\min}, \delta_{\max}]$.

The resulting generator encapsulates the initial Kriging model fit and can be used to create an arbitrary number of test function variants by randomly perturbing the θ model parameter. These test function variants will share the same structure. From a software technology standpoint, the encapsulation can be achieved by representing Kriging-based test function generators as closures.

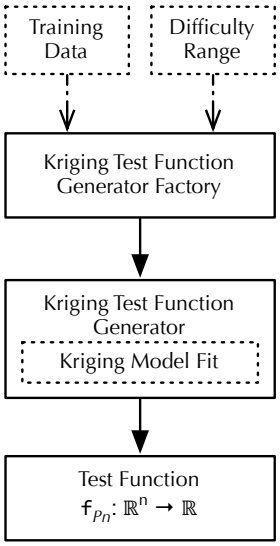


Figure 4.2: The Kriging-based test function generator. A factory function creates a Kriging test function generator based on Kriging model training data and a difficulty range $[\delta_{\min}, \delta_{\max}]$. The resulting generator encapsulates the initial Kriging model fit and can be used to create an arbitrary number of test functions by randomly perturbing the model fit.

The θ parameter occurs in the Kriging basis functions, which are of the form

$$\psi^{(i)} := \exp\left(-\sum_{j=1}^{k} \theta_j \, |x_j^{(i)} - x_j|^{p_j}\right).$$

Here, k denotes the input data dimension, i.e., the number of independent variables. Note that the Kriging basis function is a generalization of the simpler Gaussian basis function, which can be retrieved by setting $p = 2$ and all components θ_j to a constant value. Forrester et al. [2008] (p. 53) describes the θ parameter as follows:

> [θ] is essentially a width parameter that affects how far a sample point's influence extends. A low θ_j means that all points will have a high correlation, with [the dependent variable] $Y(x_j)$ being similar across our sample, while a high θ_j means that there is a significant difference between the $Y(x_j)$'s θ_j. We can therefore consider θ_j as a measure of how "active" the function we are approximating is.

Examples of the test functions generated by the Kriging-based test function generator are shown in Figure 4.3. All test functions are derived from the base function *Salustowicz 1D*, described in detail as test function **P5** below, and shown as a thick black line in both plots. The left plot shows five random test functions generated from a Kriging-based test function generator set to a difficulty range of $[0.5, 0.95]$, the right plot shows five functions from a difficulty range of $[1.01, 1.1]$. Each test function is plotted with a thin line in a different shade of gray.

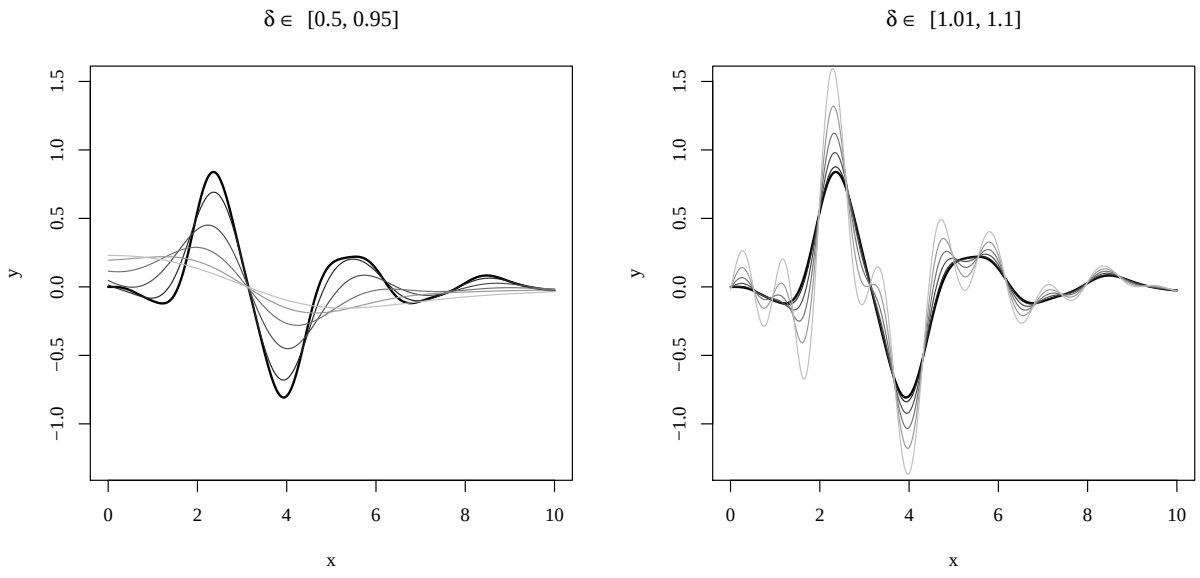


Figure 4.3: Examples of Kriging-based test functions. All test functions are derived from the base function *Salustowicz 1D* (see problem **P5**), shown as a thick black line in both plots. The left plot shows five random test functions generated from a Kriging-based test function generator set to a difficulty range of $[0.5, 0.95]$, the right plot shows five functions from a difficulty range of $[1.01, 1.1]$. Each test function is plotted with a thin line in a different shade of gray.

The following test functions were used to train the Kriging-based test function generators used in this work. As in the previous subsection, fitness cases were created by uniform random sampling in the indicated training and validation intervals.

P4 *(Air Passengers)* This test function is based on the well-known "air passengers" data set. The data consists of monthly totals of international airline passengers, from 1949 to 1960. The independent variable x is the number of months since January 1949, the dependent variable y being the number of airline passengers in thousands. The data set was first introduced by Box et al. [2008].[13] This test function has been included as a simple time series forecasting problem.

[13] Box, G., Jenkins, G., and Reinsel, G. (2008). *Time-Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, Hoboken, NJ

The training interval is $x_{\text{train}} := [1, 108]$, the validation interval is $x_{\text{validation}} := [109, 144]$, and the number of fitness cases is fixed to $N_{\text{P4}} := 144$, based on a pre-experimental study. The GP function set used is $\{+, -, *, /, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$.

P5 *(Salustowicz 1D)* This test function, first introduced by Salustowicz and Schmidhuber [1997], has been designed to be simple to read yet challenging for typical GP systems.[14]

[14] Salustowicz, R. and Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141

$$f_{\text{P5}}(x_1) := \mathrm{e}^{-x} x^3 \cos(x) \sin(x) [\sin^2(x) \cos(x) - 1]$$

The training and validation intervals of the Salustowicz 1D test function is $x_{\text{train}} = x_{\text{validation}} := [0, 10]$, where a random sample of 50% of the data is used for training, while the remainder is used for testing. The number of fitness cases is fixed to $N_{\text{P5}} := 100$, based on results of a pre-experimental study. The GP function set used is $\{+, -, *, /, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$.

In the remainder of this work, the term *problem class* is used to refer to concrete test function instances created by a test function generator. It is assumed that these instances are similar in structure and that these similarities approximate similarities found in real-world data from the same domain and same underlying generating process. Among other things, the next chapter examines whether this intra-class similarities lead to similar tuned GP system parameter settings.

4.4 Conclusions

This chapter introduced SPO and its implementation SPOT, as well as two different classes of scalable randomized test functions. The statistical methods underlying SPO, i.e., design of experiments, model-based parameter optimization, and sensitivity analysis, combined with scalable randomized test functions, will be highly valuable during the empirical analysis of modern GP algorithms carried out in the next chapter.

5
Optimizing Genetic Programming Parameters

The goal of this chapter is to provide a thorough and statistically well-founded experimental analysis of the parameter space of a modern GP system.[1] There are at least three important applications for the results of this analysis:

1. The relative effect of each GP system parameter and component on GP system performance, i.e., expected solution quality, will be exposed. Further research and development can then focus on the most important components.
2. Near-optimal parameter settings for each GP system parameter will be reported. Owing to the parameter tuning methodology used, these settings ought to be robust within a problem class, and can be seen as good default settings for that class.
3. The methodology introduced here can be easily applied to the analysis of newly proposed GP algorithm components, furthering reproducibility in GP research.

[1] Data and R source code for all experiments presented in this chapter are available at https://rsymbolic.org/projects/ofdiss/repository/revisions/master/show/experiments/rgpSpo.

The methodology applied in the analysis has been introduced in Chapter 4. Additional techniques and terms will be defined where needed. To our knowledge, this is the first application of state-of-the art sequential parameter optimization to a state-of-the art multi-objective symbolic regression system. Analysis of the results will help to validate the benefits of recent advances in the GP field, such as multi-objective selection operators, dynamic age layering, and scaled error measures[2].

[2] e.g. SMSE as introduced in Section 2.5.3

The remainder of this chapter is structured as follows: After a short overview on previous work in GP system parameter analysis and tuning, research questions for the analysis will be introduced as scientific claims. These claims will be broken down to testable statistical hypotheses. Next, an overview of the GP system parameters included in this study will be given. The parameter space will be screened in a first round of experiments and the relative importance of each parameter will be reported. Next, a method for selecting near-optimal GP function sets is presented. After that, it will be shown how the most influential parameters can be tuned by SPO, resulting in near-optimal parameter settings for a given problem class. This chapter concludes with a summary description

of the procedure proposed and best practices for selecting near-optimal GP parameter settings and GP function sets for a given problem class.

5.1 Previous Work

As EAs in general, GP is relatively robust to its parameter settings, meaning that usable results can be reached even with basic algorithms and default parameter settings based on coarse rules of thumb. Traditionally in the field of GP, the crossover rate is often set to 0.9, and the population size is chosen to be as large as possible while allowing for about 10 to 50 generations in the available compute time budget.[3] Due to its relative robustness regarding parameter settings and relative high computational effort per run, GP parameter tuning is still not common practice and the literature is quite sparse.

[3] Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza)

There are two general approaches to setting EA and GP parameters: Parameter tuning and parameter control. Parameter control is done online, i.e., during a single evolution run. Parameter control tries to gauge the effect of each controlled parameter by measuring algorithm performance in response to changes and adapts parameters based on this information. In contrast, parameter tuning is done a priori, and parameter values are kept constant during a single GP run. This work focuses on parameter tuning. Eiben et al. [1999] provides a comprehensive introduction to parameter control in EAs.[4]

[4] Eiben, A., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141

Although parameter control and parameter tuning are distinct paradigms with different terminology, different techniques, and largely separate research communities, conceptually, every parameter tuning technique can be transformed to a parameter control technique. To do so, one or multiple parameter tuning runs are interleaved with the main GP run, sharing the same compute time budget. After each parameter tuning step, the main GP run is resumed using the best parameter settings found.

The first to use statistical methods for the design and analysis of experiments (DoE) for GP were Feldt and Nordin [2000]. They employed fractional factorial designs to analyze the effect of different parameter settings to the commercial GP system Discipulus, which has been introduced shortly in Section 3.1. Discipulus does neither support multi-objective selection nor dynamic age-layering. Their study included 17 factors with two levels each. Validation set hit-rate results from three binary classification problems hinted at population size followed by number of generations as the most significant parameters. Nine of the 17 parameters had a very small or statistically insignificant effect on result quality, i.e., classifier hit rate. These results support the notion that GP systems are robust to different parameter settings only partly, as at least two parameters had a large effect on result quality.[5]

[5] Feldt, R. and Nordin, P. (2000). Using factorial experiments to evaluate the effect of genetic programming parameters. In Poli, R. et al., editors, *Proceedings of the European Conference on Genetic Programming (EuroGP 2000)*, volume 1802 of *Lecture Notes in Computer Science*, pages 271–282, Berlin. Springer

Piszcz and Soule [2006a] introduced an artificial test problem

of variable difficulty to test for a correlation between the optimal mutation rate and problem difficulty. They used simple sweeps of the mutation rate parameter for their experiments. Results show that the range of the optimal mutation rates is inversely proportional to problem difficulty, defined as solution tree depth. As an additional result, they showed that that the range of good mutation rates narrows with increasing problem difficulty.[6]

Lasarczyk [2007] successfully employed SPO to analyze and tune parameters of an algorithmic chemistry. He also tuned the parameters of a single-objective GP system using register machine programs as genotype representation (linear GP) to provide a baseline for his results. Both GP variants were tuned based on three artificial test problems. Results show that SPO is effective in identifying good parameter settings for both GP variants studied. For both GP variants, the tuned parameter settings were similar for all test problems. Some of the tuned settings lied at the boundary of their region of interest, though.[7]

White and Poulding [2009] used a DoE methodology to access the relative importance of crossover and mutation operators in the GP system ECJ that has been introduced in Section 3.1. They used a set of six test problems, including symbolic regression, classification, and control problems. Results show that crossover significantly outperformed mutation in only two out of six test problems. Their work also demonstrated the effectiveness of using DoE based on full factorial designs in tuning GP system parameters by significantly improving ECJ's default parameter settings for three out of six test problems.[8]

The work of De Lima et al. [2010] improved on Feldt and Nordin [2000]'s method by using full factorial designs to determine influential parameters as well as relevant parameter interactions of the lil-gp system, a simpler precursor of ECJ. One-factor experiments and linear regression were used to fine-tune the most influential GP parameters on two fixed test problems, binary classification and symbolic regression. The study included the six parameters population size, generations, maximum tree depth, crossover-, mutation-, and reproduction-rate. Results show that the maximum tree depth is the most important parameter, followed by number of generations and population size.[9]

[6] Piszcz, A. and Soule, T. (2006a). Genetic programming: Analysis of optimal mutation rates in a problem with varying difficulty. In Sutcliffe, G. C. J. and Goebel, R. G., editors, *Proceedings of the 19th International Florida AI Research Society Conference (FLAIRS 2006)*, pages 451–456, Menlo Park, CA. AIII Press

[7] Lasarczyk, C. W. G. (2007). Genetische Programmierung einer algorithmischen Chemie. Dissertation, TU Dortmund, Germany

[8] White, D. R. and Poulding, S. (2009). A rigorous evaluation of crossover and mutation in genetic programming. In Vanneschi, L. et al., editors, *Proceedings of the 12th European Conference on Genetic Programming (EuroGP 2009)*, volume 5481 of *Lecture Notes in Computer Science*, pages 220–231, Berlin. Springer

[9] De Lima, E. B., Pappa, G. L., de Almeida, J. M., Goncalves, M. A., and Meira, W. (2010). Tuning genetic programming parameters with factorial designs. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010)*, Piscataway, NJ. IEEE Press

5.2 *Research Questions*

There are three high-level scientific claims examined in this work. For the first, TinyGP is considered as a baseline algorithm implementing only the most basic features of GP. A common belief in the GP community can be stated as follows:

Scientific Claim 1 (C-1) *Complex problems require complex algorithms.*

Or, stated differently: TinyGP can solve tiny problems whereas hard problems require more complex algorithms.

From a naive point of view, Claim 1 goes without saying—one may simply consider that the opposite assumption is true. Therefore, it can be taken as a guideline for performing experiments and present results that are based on solid scientific and statistical assumptions.

In order to perform a sound statistical analysis, a hierarchy of complex (hard) problems will be defined. Scalable test functions based on spurious variables, as introduced in Section 4.3.1, are well suited for this goal: Increasing the number of spurious variables should decrease the success rate of the GP system.

A reference GP implementation should consolidate all essential features of GP systems in a simple and understandable manner. The TinyGP system is considered as an ideal candidate for a baseline algorithm, because it is well-known, easy to obtain, and easy to implement. RGP's GMOGP search heuristic is employed as a reference GP implementation of a modern GP algorithm. It includes at least multi-objective selection as a defining feature of state-of-the-art GP systems. Also, it is easy to obtain, relatively easy to understand, and relatively easy to implement.

THE second claim examined in this work is about parameter tuning:

Scientific Claim 2 (C-2) *The parameter settings of modern* GP *systems can be effectively tuned.*

As already mentioned, modern GP systems are GP systems incorporating multi-objective selection. As all GP systems studied in the previous work described in Section 5.1 did not implement multi-objective selection, it has to be shown that parameter tuning is effective for modern GP systems. By breaking down this claim to testable hypotheses, detailed information about the relative importance of parameters and their near-optimal settings for a given GP task should be obtainable.

Assuming that the parameters of modern GP systems can be effectively tuned at all, the question about parameter setting robustness quickly arises. If tuned parameter settings are highly specific to a single GP problem, parameter tuning would be of limited utility. It is therefore claimed that:

Scientific Claim 3 (C-3) *Tuned* GP *system parameter settings are robust within a problem class.*

If this claim can be backed by empirical evidence, tuned parameter settings can be assumed useful at least within a larger class of GP tasks, e.g. different symbolic regression tasks from the same problem domain.

5.3 GP System Parameter Overview

This section introduces the GP system parameter space analysed in this study. First, parameters common to all GP search heuristics are

explained. Next, parameters specific to the TinyGP and GMOGP search heuristics are reproduced for convenience. These parameters have been already explained in detail in Section 2.9.

5.3.1 Common Parameters

Table 5.1 shows GP system parameters common to all search heuristics with their respective domains, types, and defaults. In all experiments, default values where used unless a parameter is subject to screening or tuning. For notational brevity, error measures and function sets are referred to by index number in the experiment designs described later.

	Variable (Symbol)	*Domain*	*Default*
Crossover Probability	`crossoverP` ($p_{\text{crossover}}$)	$[0,1]$	0.5
Constant Mutation Weight	`constantMutationPw` (p_{mconst})	$[0,1]$	0
Constant Mut. Mean	`constantMutationMu`	$\mathbb{R}$	0
Constant Mut. SD	`constantMutationSigma`	$\mathbb{R}$	1
Function Mutation Weight	`functionMutationPw` (p_{mfunc})	$[0,1]$	0
Subtree Mutation Weight	`subtreeMutationPw` (p_{msubtree})	$[0,1]$	1
Individual Size Limit	`individualSizeLimit`	$\mathbb{N}$	64
Error Measure Number	`errorMeasureNum` (i_{error})	$\{1,\ldots,4\}$	1
Function Set Number	`functionSetNum` (i_{funcset})	$\{1,\ldots,4\}$	4

Table 5.1: Parameters of the RGP system independent of the search heuristic.

Table 5.2 presents index numbers for error measures available in RGP by default and used in this study, where index numbers for function sets used in this study are presented in Table 5.3.

i_{error}	*Error Measure*	*Description*
1	SMSE	Scaled mean squared error
2	RMSE	Root mean squared error
3	SSE	Sum squared error
4	MAE	Mean absolute error

Table 5.2: Default error measures available in RGP and used in this study.

i_{funcset}	*Function Set*
1	$\{+,-,\times,\div\}$
2	$\{+,-,\times,\sin,\exp\}$
3	$\{+,-,\times,\div,\sin,\exp,\log,\sqrt{x}\}$
4	$\{+,-,\times,\div,\sin,\cos,\tan,\exp,\log,\sqrt{x}\}$

Table 5.3: Default function sets available in RGP and used in this study.

5.3.2 TinyGP Parameters

Table 5.4 shows GP system parameters specific to the TinyGP search heuristic.

5.3.3 GMOGP Parameters

GP system parameters specific to the GMOGP search heuristic are shown in Table 5.5. The selection criteria genotypic complexity and

	Variable (Symbol)	*Domain*	*Default*
Population Size	`mu` (μ)	$\mathbb{N}$	300
Tournament Size	`tournamentSize` ($s_{\text{tournament}}$)	$\mathbb{N}$	2
Crossover Probability	`crossoverP` ($p_{\text{crossover}}$)	$[0,1]$	0.9

Table 5.4: Parameters of the TinyGP search heuristic.

genotypic age can be enabled and disabled individually, yielding three valid configurations of the GMOGP search heuristic:

- single-objective selection based on goodness of fit (GMOGP-F)
- multi-objective selection based on goodness of fit and individual age (GMOGP-FA)
- multi-objective selection based on goodness of fit and individual complexity (GMOGP-FC)
- multi-objective selection based on goodness of fit, individual age, and genotypic individual complexity (GMOGP-FCA)

Depending on the configuration selected, other algorithm components may be disabled. Explicit diversity preservation through Age-Fitness Pareto Optimization is available only when the individual age criterion is enabled.

To handle constraints of the parameters λ and ν relative to μ, these are transformed to box-constrained relative parameters $\lambda_{\mu\,\text{rel}}$ and $\nu_{\mu\,\text{rel}}$ according to the following equations:

$$\begin{aligned} \lambda &= \max(2, \lceil \lambda_{\mu\,\text{rel}} \cdot \tfrac{\mu}{2} \rceil) \\ \nu &= \lceil \nu_{\mu\,\text{rel}} \cdot \mu \rceil \end{aligned} \tag{5.1}$$

Note that with this transformation, the minimum value for λ is 2, i.e., at least two children will always be created, avoiding a configuration resulting in purely random search.

	Variable (Symbol)	*Domain*	*Default*
Population Size	`mu` (μ)	$\{8,\ldots,256\}$	100
Children per Generation (relative)	`lambdaRel` ($\lambda_{\mu\,\text{rel}}$)	$[0,1]$	1
New Individuals per Generation (relative)	`nuRel` ($\nu_{\mu\,\text{rel}}$)	$[0,1]$	0.5
Age Layering	`ageLayering` (b_{age})	$\mathbb{B}$	`true`
Parent Selection Probability	`parentSelectionP` (p_{psel})	$[0,1]$	1

Table 5.5: Parameters of the GMOGP search heuristic, transformed to remove linear constraints between parameters.

5.4 Pre-Experimental Planning

Following the experimental framework presented by Bartz-Beielstein [2006], a sound experiment setup requires the specification of the 1) optimization problem, 2) performance measure, 3) initialization method, and 4) termination method. [10]

[10] Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation—The New Experimentalism*. Natural Computing Series. Springer, Berlin

5.4.1 Optimization Problem

The symbolic regression test functions used in this study have two components: 1) a test function, say f, and 2) a measure, which determines the distance between the true function f and its GP approximation, say $\hat{f}$.

Test functions where presented in Section 4.3. RMSE is the performance measure used in the comparisons. See Section 2.5.3 for a definition and details. GP runs where performed based on *training error* calculated on the training sets described in Section 4.3.

During the pre-experimental planning phase, a large number of function evaluations n_0, the so-called *budget*, has been used the determine an adequate percentage of GP runs that reach a predefined (small) distance to the optimum. *Run-length distributions* (RLDs), as introduced by Parkes and Walser [1996], are generated to estimate a suitable budget $n \leq n_0$ for the actual experiments.[11]

To avoid floor and ceiling effects, an adequate problem design, e.g., number of function evaluations as well as reasonably chosen training, validation, and test sets, has been determined.

[11] Parkes, A. J. and Walser, J. P. (1996). Tuning local search for satisfiability testing. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI 1996)*, pages 356–362, Menlo Park, CA. AIII Press

5.4.2 *Performance Measure*

To avoid overfitting on the training set, the *test error*, i.e., the expected prediction error over an independent test sample x_t, is preferred. The comparison of the algorithm designs is based on these test-set data. To enable fair comparisons and to avoid floor and ceiling effects, RLDs of the TinyGP implementation are generated.

5.4.3 *Initialization and Termination Method*

Classical ramped half-and-half initialization, as introduced in Section 2.6.1 was used in all experiments. To determine suitable termination methods, RLDs were generated. The number of solution evaluations, i.e., the number of fitness cases times the number of test function evaluations, was chosen as the termination criterion.

5.4.4 *Compute Time Budget Calibration*

During the first stage of experimentation, no parameter tuning will be performed. The goal is to discover positive correlations between algorithm complexity and problem difficulty using default algorithm parameter settings.

First experiments were set up to calibrate the reference algorithm TinyGP to the problem.

Statistical Hypothesis 1 (H-1) *TinyGP requires more function evaluations to reach the same success rate as problem difficulty increases.*

RLDs are suitable means to measure performance and to determine an adequate budget. This is necessary because floor and ceiling effects should be avoided. A large number of function evaluations, i.e., a budget of $n_0 = 1e6$, was chosen to generate the RLDs. The set of test functions consists of $\{f_{P1}, f_{P2}, f_{P3}\}$.

The investigation of H-1 has two significant results: First, it is ensured that the reference algorithm is able to solve this problem (at least it should be able to improve an existing candidate solution).

Additionally, the correct number of test function evaluations for comparisons is determined.

Second, test functions which are far too easy (or too hard) for the reference algorithm are detected. Further considerations are needed in the case that the reference algorithm is unable to solve a test function.

Next, we introduce GMOGP-F as a base variant of the new algorithm to be analyzed.

Statistical Hypothesis 2 (H-2) GMOGP-F *is competitive with the reference algorithm.*

To analyze H-2, the same setup as for H-1 was used.

These two series of experiments belong to the pre-experimental planning phase, because they are needed to set up fair comparisons. If either H-1 or H-2 has to be rejected, the experiment setup should be reconsidered, e.g., the set of test function should be modified or the computational budget should be increased.

Calibration Results To generate the experimental data needed to test the statistical hypotheses H-1 and H-2, experiments for each hypothesis where prepared in the RGP framework. These experiments where then executed on a 48 core compute cluster. Generation of result reports and visualizations where automated by the RGP framework.

Statistical Hypothesis 1 (H-1) Table 5.6 and Figure 5.1 show that TinyGP's success rates decrease as problem complexities increase. As the algorithm can get stuck in local optima and basing RLD calculation on test data instead of training data adds additional noise, this decrease is not strictly monotonic. Nonetheless, the data does not indicate that H-1 has to be rejected. During pre-experimental planning, the Sine Cosine test problem (P3) proved too difficult for the reference algorithm (TinyGP), and was therefore excluded from further experiments.

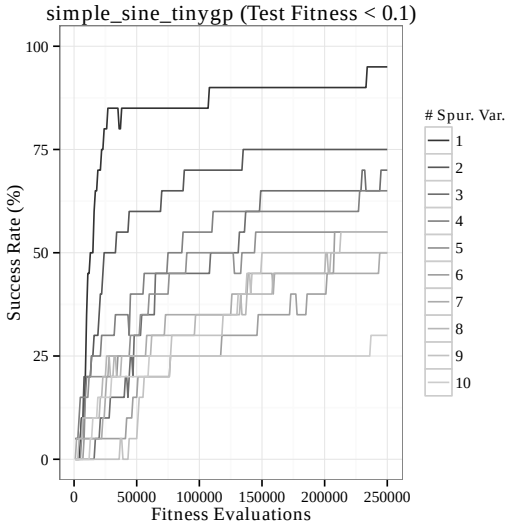
Statistical Hypothesis 2 (H-2) On both test functions studied, GMOGP-F shows significantly better performance than the reference, as visible in Table 5.6 and Figure 5.1. Hypothesis H-2 does not have to be rejected. This concludes the analysis of the pre-experimental planning phase.

5.5 GP System Parameter Screening

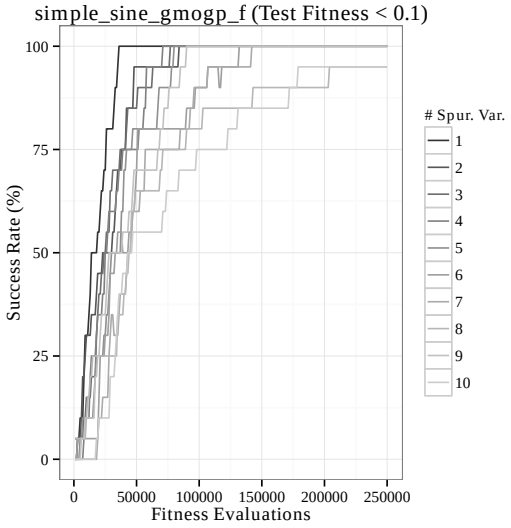
The goal of the next series of experiments is twofold. First, the most important GP system parameters, i.e., the parameters with the largest effect on GP performance, should be identified. Second, the size of the parameter space should be reduced either by removing parameters of negligible effect or by fixing influential parameters to

		Difficulty (# Spurious Variables)									
Search Heuristic	*Test Function*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
TinyGP	Simple Sine (P1)	95	75	70	65	55	55	50	50	55	30
	Newton Problem (P2)	80	80	60	70	60	35	40	45	35	10
	Sine Cosine (P3)	10	20	20	15	10	0	5	15	5	5
GMOGP-F	Simple Sine (P1)	100	100	100	100	100	100	100	95	100	95
	Newton Problem (P2)	90	90	90	80	50	75	65	55	40	30

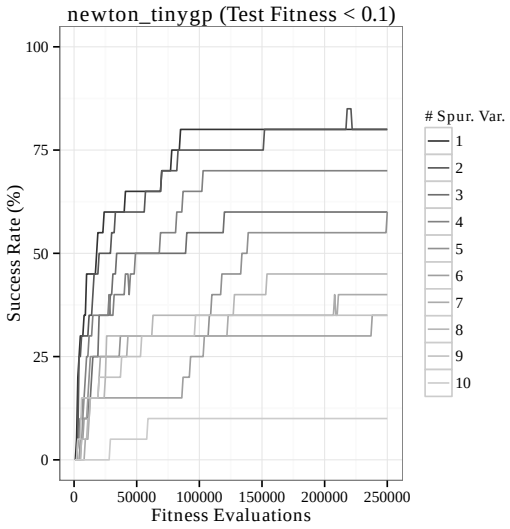
Table 5.6: Run length distributions (%) for TinyGP and GMOGP-F at 250,000 fitness function evaluations.



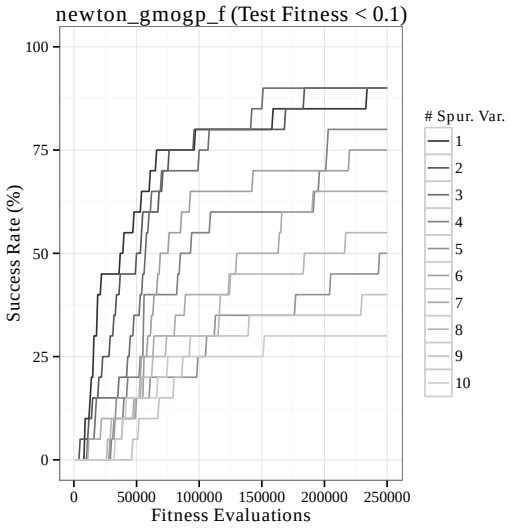
(a) Simple Sine (P1), TinyGP search heuristic



(b) Simple Sine (P1), GMOGP-F search heuristic



(c) Newton Problem (P2), TinyGP search heuristic



(d) Newton Problem (P2), GMOGP-F search heuristic

Figure 5.1: RLD plots for TinyGP and GMOGP-F. Algorithm success rates are based on a fitness threshold of 0.1. Each plot shows ten difficulty levels (number of spurious variables) of a single test function for a single search heuristic, based on 20 independent runs for each combination of search heuristic, test function, and difficulty.

universally good settings. This size reduction leads to a GP system that is easier to use, as less parameters have to be set, as well as faster and more efficient to tune.

5.5.1 Experiment Setup

From now on, the GMOGP-FCA search heuristic will be used, i.e., GMOGP with multi-objective selection based on goodness of fit, individual age, and genotypic individual complexity. The previous experiments where needed to estimate realistic run time budgets. Also, the set of test functions is changed to $\{f_{P4}, f_{P5}\}$, which are more challenging for symbolic regression and therefore more relevant to more advanced search heuristics. In the following parameter screening experiments, fixed test functions are used, i.e., the difficulty of the Kriging test function generator has been set to the fixed value of 1.0. See Section 4.3 for details. Randomized test functions will be introduced in the next series of experiments on GP function set selection.

Optimization Problem As GMOGP-FCA is multi-objective search heuristic, the definition of the optimization problem has to be extended by adding additional objectives. In addition to goodness of fit measured as described in the last section, these objective are genotypic individual complexity and individual age. See Section 2.9.3 for detailed definitions.

Initialization and Termination Method Initial experiments showed that different parameter settings can lead to significant differences in the runtime required to perform a fixed number of fitness evaluations in the GMOGP-FCA search heuristic. This effect can be clearly seen in Figure 5.2 and explained by the fact that, depending on parameter settings, GP operators have to be applied more or less often to reach the same number of fitness function evaluations. For example, with small population sizes (μ), the selection operator will be applied more often than with large μ for a fixed number of fitness function evaluations.

In most real-world GP applications, only a fixed compute time budget is available. Therefore, from now on, the expiration of a fixed compute time budget was chosen as the termination criterion. The compute time budget has been calibrated to 12 hours on a single thread on an Intel Xeon E5530 CPU running at 2.4 GHz. With this budget, acceptable results are reached with GMOGP-FCA default parameter settings on both P4 and P5. For illustration, Figure A.1 in Appendix A visualizes the results for test function P5 obtained by 12 calibration runs with a budget of 12 hours each. The results for test function P4 are of comparable quality.

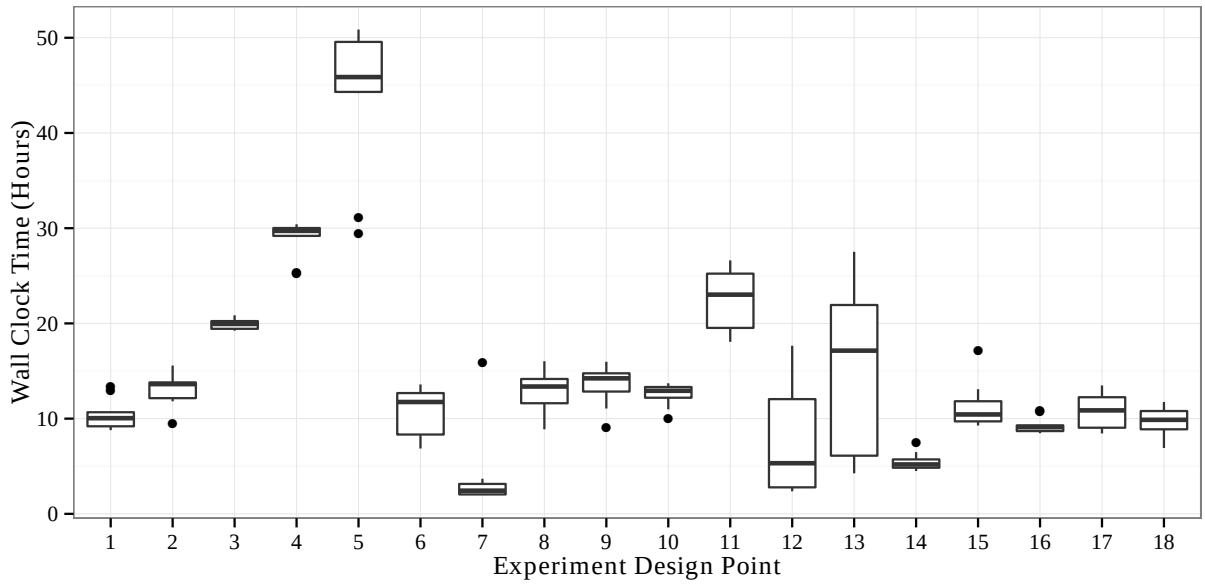


Figure 5.2: Wall clock time taken for ten million fitness evaluations using GMOGP-FCA on Salustowicz 1D (P5), for different algorithm parameter settings (experiment design points) with ten runs each. Measurements were taken on a single thread on an Intel Xeon E5530 CPU running at 2.4 GHz.

5.5.2 *Statistical Hypotheses*

The experiments performed in this section are designed to test the following three statistical hypotheses.

Statistical Hypothesis 3 (H-3) *There exist parameters that have a significant effect on* GMOGP-FCA *system performance as measured by validation* MAE *of the best solution found.*

This hypothesis states that there are GMOGP-FCA system parameters that have a statistically significant effect on the system's performance. If this hypothesis has to be rejected, the experiment setup should be reconsidered. Otherwise, the following hypotheses can be tested. This approach is related to standard procedures in analysis of variance (ANOVA) .

Statistical Hypothesis 4 (H-4) GMOGP-FCA *system parameters show significant differences in their effect on* GMOGP-FCA *system performance as measured by validation* MAE *of the best solution found.*

Some parameters are expected to have a larger effect on the GP system's performance than others. If this hypothesis has to be rejected, a reduction of the parameter space dimensionality cannot be justified.

Statistical Hypothesis 5 (H-5) *The choice of the* GP *function set has a significant effect on* GMOGP-FCA *system performance as measured by validation* MAE *of the best solution found.*

It is expected that the choice of the GP function set parameter i_{funcset} should exert a significant effect on the GP system's performance. Depending on the standpoint of the user, this parameter

can be considered as an algorithm parameter or as a problem parameter. The function set defines in large parts the set of possible solutions as well as the size of the genotype set, therefore being an important factor in overall problem difficulty. This intuitive argument is formalized by hypothesis H-5.

5.5.3 *Screening Design*

To test the statistical hypotheses H-3 to H-5 introduced in the previous subsection, experiments based a Plackett-Burman Screening Design are conducted. See Grönmping [2014] for details on this type of experiment design and the R package FrF2 that implements it.[12] Table 5.7 shows the design used. For each of the 18 different parameter settings (design points), ten independent GP runs where executed on test functions P4 and P5, leading to a total of 360 GP runs with a compute time budget of 12 hours each.

[12] Grönmping, U. (2014). R package FrF2 for creating and analyzing fractional factorial 2-level designs. *Journal of Statistical Software*, 56(1):1–56

	i_{funcset}	p_{mconst}	$p_{\text{crossover}}$	b_{age}	i_{error}	p_{mfunc}	$\lambda_{\mu\,\text{rel}}$	μ	$\nu_{\mu\,\text{rel}}$	p_{psel}	p_{msubtree}
1	2	0.5	0.5	`true`	2	0.5	0.5	132	0.5	0.5	0.5
2	2	0.5	0.5	`true`	2	0.5	0.5	132	0.5	0.5	0.5
3	1	1.0	0.0	`true`	4	0.0	1.0	256	1.0	0.0	0.0
4	1	0.0	1.0	`false`	4	1.0	0.0	256	1.0	1.0	0.0
5	1	0.0	0.0	`true`	1	1.0	1.0	8	1.0	1.0	1.0
6	4	1.0	1.0	`false`	1	0.0	1.0	8	1.0	1.0	0.0
7	1	1.0	1.0	`false`	4	1.0	1.0	8	0.0	0.0	1.0
8	4	0.0	0.0	`false`	4	0.0	1.0	256	0.0	1.0	1.0
9	2	0.5	0.5	`true`	2	0.5	0.5	132	0.5	0.5	0.5
10	2	0.5	0.5	`true`	2	0.5	0.5	132	0.5	0.5	0.5
11	1	1.0	1.0	`true`	1	0.0	0.0	256	0.0	1.0	1.0
12	1	0.0	0.0	`false`	1	0.0	0.0	8	0.0	0.0	0.0
13	4	0.0	1.0	`true`	1	1.0	1.0	256	0.0	0.0	0.0
14	4	1.0	0.0	`true`	4	1.0	0.0	8	0.0	1.0	0.0
15	4	0.0	1.0	`true`	4	0.0	0.0	8	1.0	0.0	1.0
16	4	1.0	0.0	`false`	1	1.0	0.0	256	1.0	0.0	1.0
17	2	0.5	0.5	`true`	2	0.5	0.5	132	0.5	0.5	0.5
18	2	0.5	0.5	`true`	2	0.5	0.5	132	0.5	0.5	0.5

Table 5.7: Plackett-Burman Screening Design for the GMOGP-FCA search heuristic. For each design point, ten independent GP runs where executed on test functions P4 and P5, resulting in a total of 360 runs with a compute time budget of 12 hours each.

5.5.4 *Screening Results*

Descriptive statistics of the raw result data for the parameter screening experiments are reproduced in Table 5.8 (test function P4) and Table 5.9 (test function P5). Each of the 18 design points corresponds to a set of ten result Pareto fronts. For each result Pareto front, the solution with absolute lowest validation MAE (best) and the solution at the Pareto front knee is reported in the tables. For both best and knee solutions, medians and standard deviations (SD) for validation MAE, and genotypic complexity are reported. For best solutions, RLD values are also reported to simplify result interpretation. For P4 (P5), a best solution validation MAE of 60 (0.05) has been used as RLD threshold.

Here, the knee of the Pareto front is defined as the Pareto front member with minimum normalized euclidean distance to the

	MAE (best)			Complexity (best)		MAE (knee)		Complexity (knee)	
	RLD (%)	Median	SD	Median	SD	Median	SD	Median	SD
1	0.70	35.66	9.31	347.00	546.69	51.47	9.87	47.00	500.06
2	0.90	39.61	7.27	348.50	652.74	51.36	4.43	37.00	22.93
3	1.00	50.91	0.08	11.00	75.82	50.92	0.02	11.00	0.00
4	0.80	53.29	4.88	112.00	287.37	62.15	8.74	11.00	10.72
5	0.00	212.70	119.52	421.00	318.12	410.62	6.20	1.00	0.00
6	0.00	221.22	144.44	44.00	451.04	411.74	82.81	1.00	185.23
7	0.10	292.98	121.29	1.00	9.11	292.98	85.45	1.00	8.64
8	0.40	65.67	37.83	100.00	117.20	140.34	32.52	8.00	1.45
9	1.00	35.55	8.83	728.50	575.41	51.08	5.68	32.00	37.38
10	0.70	37.83	8.11	506.00	576.59	55.06	110.54	16.50	12.02
11	0.10	292.96	94.45	3.00	226.99	292.98	70.92	1.00	1.93
12	0.00	292.98	4475.94	1.00	12.58	292.98	4475.94	1.00	12.58
13	0.20	93.05	82.18	64.50	172.24	292.98	0.00	1.00	0.00
14	0.00	292.98	48.70	1.00	16.60	292.98	43.44	1.00	2.54
15	0.30	62.58	5.58	666.50	958.94	65.21	32.67	15.00	603.40
16	0.00	247.85	67.13	57.00	24.92	408.20	53.72	1.00	8.37
17	0.80	36.49	9.51	869.50	776.84	50.99	6.55	59.50	202.35
18	0.90	39.15	8.45	826.00	519.26	51.62	4.46	20.50	200.07

Table 5.8: Parameter screening results for the GMOGP-FCA search heuristic, Air Passengers 1D test function (P_4). The RLD threshold was set to a validation MAE of 60.

utopia point. Given a matrix M denoting a Pareto front, where each row denotes a Pareto front member and each column the vector of normalized objective values of a single optimization objective, the utopia point is the point consisting of the row minima of M.

	MAE (best)			Complexity (best)		MAE (knee)		Complexity (knee)	
	RLD (%)	Median	SD	Median	SD	Median	SD	Median	SD
1	0.90	0.03	0.03	1326.50	640.84	0.06	0.02	166.00	67.33
2	1.00	0.02	0.01	1107.50	546.95	0.05	0.02	152.00	84.32
3	0.00	0.14	0.01	506.00	145.01	0.38	0.03	1.00	0.00
4	0.00	0.15	0.01	573.00	239.98	0.30	0.04	1.00	41.71
5	0.00	0.24	0.45	546.00	301.00	1.41	1.08	1.00	95.59
6	0.00	0.32	0.24	165.00	2427.91	0.71	0.96	1.00	63.26
7	0.00	0.29	1.02	5.00	43.87	0.84	0.95	5.00	36.03
8	0.00	0.14	0.02	117.00	89.08	0.16	0.02	54.00	10.41
9	1.00	0.02	0.01	1388.50	512.75	0.06	0.02	128.00	62.38
10	1.00	0.02	0.01	1392.00	603.52	0.04	0.01	184.50	72.64
11	0.00	0.20	1.32	384.00	301.15	1.25	2.22	1.00	12.65
12	0.00	5.85	13.67	1.00	294.15	6.40	12.93	1.00	12.58
13	0.00	0.13	0.06	289.50	1837.87	0.19	2.01	31.50	383.51
14	0.00	0.24	0.13	6.00	27.28	0.48	2.27	3.00	16.37
15	0.60	0.04	0.05	1889.00	2483.50	0.11	0.04	49.00	356.08
16	0.00	0.12	0.05	27.00	59.89	0.42	0.15	9.00	4.65
17	0.90	0.02	0.03	1169.00	601.91	0.05	0.03	156.00	71.93
18	0.90	0.03	0.02	1153.00	852.38	0.05	0.04	146.50	97.62

Table 5.9: Parameter screening results for the GMOGP-FCA search heuristic, Salustowicz 1D test function (P_5). The RLD threshold was set to a validation MAE of 0.05.

Statistical Hypothesis 3 (H-3) To test the hypothesis that at least some GMOGP-FCA system parameters have a significant effect on GMOGP-FCA system performance as measured by validation MAE of the best solution found, a Kruskal-Wallis rank sum test[13] is performed for each parameter on test functions P_4 and P_5. The null hypothesis of this test is that the location parameters of the

[13] Kruskal, W. and Wallis, W. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621

distribution of the validation MAE values are the same for each parameter setting. The test's p-value denotes the probability for the input data being observed, assuming the null hypothesis being true.

The Kruskal-Wallis test tests whether the mean ranks of the dependent variable (here: the validation MAE values) are the same for all parameter settings. As the test is non-parametric, the dependent variable does not need to be normally distributed.

Table 5.10 summarizes the results of the Kruskal-Wallis test. For both test functions and all parameters, the p-values reported are very small. Hence, the null hypothesis can be rejected. All parameters exert a statistically significant effect on the best solution validation MAE values. The only exception being the *repeat number*, a pseudo-parameter in the range $\{1, 2, \ldots, 10\}$ identifying a specific GP run among the ten repeats for each design point. This parameter has been included to validate the test results, as it should be irrelevant for the resulting validation MAE value. Now, we can proceed with testing statistical hypothesis 4 (H-4).

Table 5.10: Kruskal-Wallis rank sum test performed on the best individual validation MAE values of the parameter screening results for the GMOGP-FCA search heuristic. Pseudo-parameters are marked with an asterisk.

	Air Passengers 1D (P4)		Salustowicz 1D (P5)	
Parameter	χ^2	p-Value	χ^2	p-Value
i_{funcset}	76.68	2.23×10^{-17}	86.96	1.31×10^{-19}
p_{mconst}	71.86	2.49×10^{-16}	89.49	3.68×10^{-20}
$p_{\text{crossover}}$	70.85	4.13×10^{-16}	85.08	3.36×10^{-19}
b_{age}	70.72	4.41×10^{-16}	87.01	1.27×10^{-19}
i_{error}	71.35	3.21×10^{-16}	86.07	2.04×10^{-19}
p_{mfunc}	75.43	4.18×10^{-17}	91.79	1.17×10^{-20}
$\lambda_{\mu\,\text{rel}}$	71.29	3.30×10^{-16}	84.94	3.60×10^{-19}
μ	79.17	6.45×10^{-18}	87.08	1.23×10^{-19}
$\nu_{\mu\,\text{rel}}$	77.32	1.62×10^{-17}	92.81	7.04×10^{-21}
p_{psel}	71.82	2.53×10^{-16}	84.94	3.59×10^{-19}
p_{msubtree}	72.93	1.45×10^{-16}	85.79	2.35×10^{-19}
repeat number*	3.78	0.93	2.15	0.99

Statistical Hypothesis 4 (H-4) To test the hypothesis that GMOGP-FCA system parameters have significant differences in their effect on GMOGP-FCA system performance as measured by validation MAE of the best solution found, empirical attainment function (EAF) difference plots will be used. EAF difference plots are visualizations of the objective space of a multi-objective optimization problem.

For each point p in objective-space, the first-order attainment function gives the probability of a multi-objective optimization algorithm finding at least a single solution whose objective vector p' Pareto-dominates or is equal to p. This probability can be estimated from the results of multiple independent algorithm runs. It's estimator is called the (first-order) EAF. The results of GMOGP-FCA are bi-objective, hence EAFs can be efficiently calculated and easily visualized. In an EAF visualization, goodness of fit is plotted on the

x-axis, genotypic complexity is plotted on the y-axis, and EAF values are represented with different shades of gray. To compare two algorithms or, as in this case, two parameter settings for the same algorithm, the differences of two EAFs can be visualized using the same principles. EAF difference plots come in side-by-side pairs, where differences in favor of algorithm (parameter setting) A are shown in the left plot, and differences in favor of algorithm (parameter setting) B are shown in the right plot. Difference magnitudes are encoded by gray level. For more details on EAF plots and their application areas, see López-Ibáñez et al. [2010].[14]

[14] López-Ibáñez, M., Paquete, L., and Stützle, T. (2010). Exploratory analysis of stochastic local search algorithms in biobjective optimization. In Bartz-Beielstein, T. et al., editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 209–222. Springer, Berlin

Figures A.2 to A.12 in Appendix A show EAF difference plots for each parameter included in this screening. These plots are based on the experiment result data summarized in Tables 5.8 and 5.9. In addition to the EAF difference magnitudes encoded as gray levels as described above, the overall best and worst attainment surfaces for both parameter settings are plotted as solid black lines, and the median attainment surface corresponding to each of the two parameter settings as a dashed black line. EAF difference plots obtained via test function P4 are shown in the top row, plots obtained via test function P5 are shown in the bottom row.

Regarding hypothesis H-4, these plots clearly indicate significant differences between parameter effects on GMOGP-FCA system performance, measured both by validation MAE and genotypic complexity of the solutions obtained. According to the EAF difference magnitudes shown, the parameters i_{funcset}, i_{error}, and μ have a particularly strong effect (see Figure 5.3, and Figures A.2, A.3, and A.4 in Appendix A), while the parameters p_{msubtree} and b_{age} are of secondary importance, at least with the test functions studied (see Figure 5.4 and Figures A.8 and A.11 in Appendix A). Therefore, H-4 does not have to be rejected.

Statistical Hypothesis 5 (H-5) Based on the EAF difference magnitudes visible in Figure A.2, the hypothesis that the choice of the GP function set has a significant effect on GMOGP-FCA system performance as measured by validation MAE of the best solution found cannot be rejected. Note that the choice of a near-optimal function set is problem dependent. With test function P4, the best compromises between goodness of fit and genotypic complexity are found with function set number 1, whereas with test function P5, the best compromises between goodness of fit and genotypic complexity are found with function set number 4.

5.6 GP Function Set Selection

An important result of the last series of experiments was that the choice of the GP function set has a significant effect on GMOGP-FCA system performance as measured by validation MAE as well as by solution complexity. As an additional result, it was found

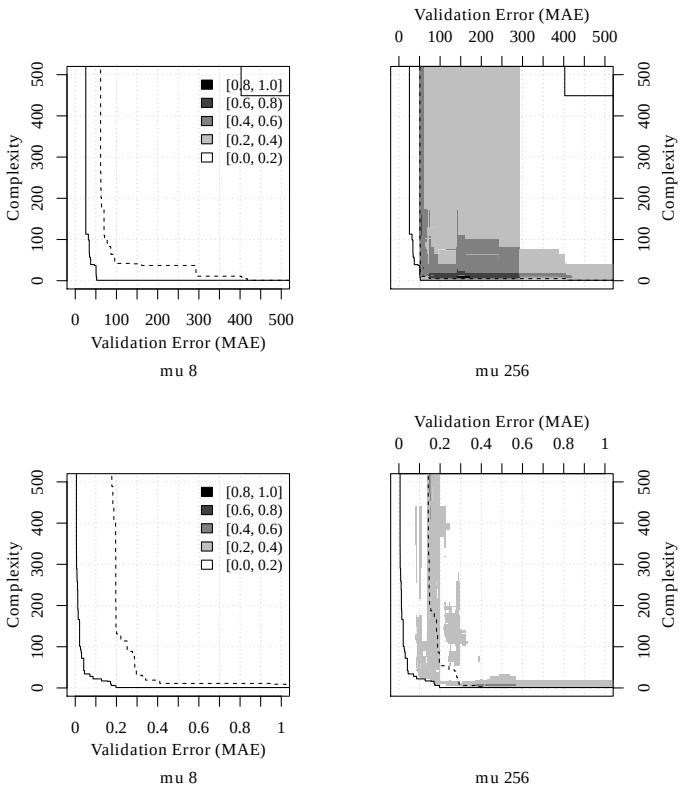


Figure 5.3: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P4, top row) and Salustowicz 1D (P5, bottom row), μ parameter. The grand-best and grand-worst attainment surfaces are shown as solid lines. The 50% attainment surface is shown as a dotted line.

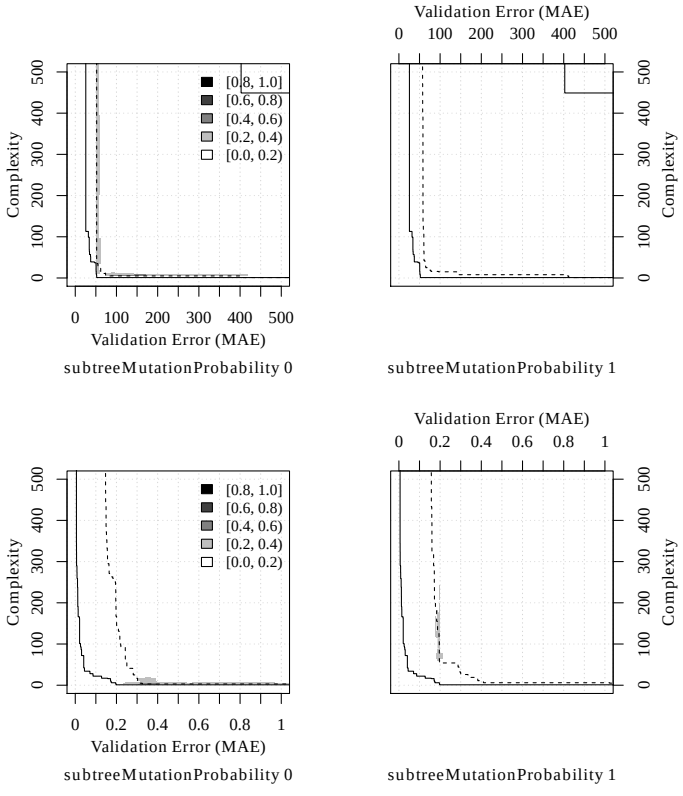


Figure 5.4: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P4, top row) and Salustowicz 1D (P5, bottom row), p_{msubtree} parameter. The grand-best and grand-worst attainment surfaces are shown as solid lines. The 50% attainment surface is shown as a dotted line.

that the choice of a near-optimal function set is dependent on the problem to be solved by GP. It seems therefore worthwhile to study GP function set selection in more depth and to develop a methodological approach to this task. This section introduces an empirical approach to GP function set selection by means of an extensive example.

5.6.1 Experiment Setup

In the next series of experiments, the same basic setup as in the previous series of experiments will be used. A larger DoE will be necessary, which forces a reduction in the compute-time budget available to each individual GP run from twelve hours to four hours. The leads to a certain loss of solution quality, but as all solutions suffer this loss, result ranks should not be affected in a major way. Result quality increases sub-linearly with increasing compute time. Visual inspection showed that on test functions P_4 and P_5, a compute time budget of two hours is sufficient to reach results of acceptable quality.

In the following GP function set selection experiments, randomized test functions are used for the first time. The difficulty range of the Kriging test function generator was set to $[0.8, 1.2]$. The test function generator has been explained in detail in Section 4.3. The base function used to train the Kriging test function generator was Salustowicz 1D (P_5). Results obtained from the following experiments are therefore valid for the *problem class* generated by the Kriging test function generator around the base function Salustowicz 1D.

To make the GP function set selection problem accessible to experimentation, it is defined as a subset-selection problem: From the set of available functions $\{+, -, \times, \div, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$, select the subset that optimizes GMOGP-FCA system performance as measured by best solution validation MAE for a given problem class.

5.6.2 Statistical Hypothesis

The experiments performed in this section are designed to test the following statistical hypothesis.

Statistical Hypothesis 6 (H-6) *Given the* GP *function set* $F := \{+, -, \times, \div, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$*, there exists a proper subset* $\hat{F} \subset F$ *that significantly improves* GMOGP-FCA *system performance on the Salustowicz 1D problem class as measured by best solution validation* MAE*.*

This hypothesis states that using a smaller and less generic GP function set appropriate to the problem class at hand significantly improves solution quality.

5.6.3 Experiment Design

As the choice of the GP function set may interact with the choice of other GMOGP-FCA system parameters, these are also included in the parameter set studied. Table 5.11 summarizes this parameter set. The presence of a GP building block function is encoded by a binary factor. The functions addition ($+$) and multiplication ($\times$) are always present in the GP function set. GMOGP-FCA parameters missing in this table were set to the default values described in Section 5.3.

	Variable (Symbol)	*Domain*
Crossover Probability	`crossoverP` ($p_{\text{crossover}}$)	$[0,1]$
Constant Mutation Weight	`constantMutationPw` (p_{mconst})	$[0,1]$
Error Measure Number	`errorMeasureNum` (i_{error})	$\{1,\ldots,4\}$
Population Size	`mu` (μ)	$\{8,\ldots,256\}$
Children per Generation (relative)	`lambdaRel` ($\lambda_{\mu\,\text{rel}}$)	$[0,1]$
New Individuals per Generation (relative)	`nuRel` ($\nu_{\mu\,\text{rel}}$)	$[0,1]$
Parent Selection Probability	`parentSelectionP` (p_{psel})	$[0,1]$
Subtraction	`FSUB` ($-$)	$\mathbb{B}$
Division	`FDIV` ($\div$)	$\mathbb{B}$
Sine	`FSIN` (sin)	$\mathbb{B}$
Cosine	`FCOS` (cos)	$\mathbb{B}$
Tangent	`FTAN` (tan)	$\mathbb{B}$
Exponential	`FEXP` (exp)	$\mathbb{B}$
Natural Logarithm	`FLOG` (log)	$\mathbb{B}$
Square Root	`FSQRT` ($\sqrt{x}$)	$\mathbb{B}$

Table 5.11: GMOGP-FCA system parameters, extended with binary factors for GP function set selection.

The parameter space under examination has five continuous parameters, two integer parameters, and eight boolean parameters, i.e., 15 parameters in total. A Latin Hypercube Design with 256 design points is generated. For each design point, four independent GP runs are executed on two test functions generated by the Kriging-based test function generator described in Section 4.3.2, leading to a total of 2048 GP runs with a compute time budget of four hours each. See Bartz-Beielstein and Zaefferer [2012] for details on this type of DoE.[15]

[15] Bartz-Beielstein, T. and Zaefferer, M. (2012). A gentle introduction to sequential parameter optimization. Cologne Open Science, Schriftenreihe CI-plus TR 01/2012, Bibliothek der Fachhochschule Köln, Betzdorfer Str. 2, 50679 Köln, Germany

5.6.4 Experiment Results

There are $2^8 = 256$ possible subsets of F, too many to study individually. To reduce complexity and generate knowledge about the properties of GP function sets well suited to the Salustowicz 1D problem class, an analysis based on regression trees will be employed.

In the first step of this analysis, a Box-Cox transformation is applied on the dependent variable for the analysis, i.e., validation MAE of the best solution found. After considering the transfor-

mations $x^{-1}, x^{-0.5}, \log x$, and $\sqrt{x}$, the square root transform was determined as most suitable to generate symmetry in the data. See Figure 5.5 for a visual comparison of the original and transformed result data.

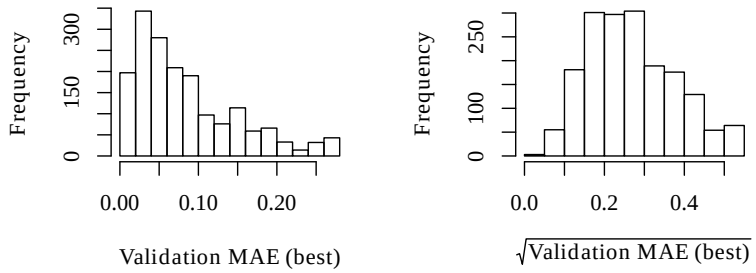


Figure 5.5: Original and transformed result data. The left plot shows a histogram of untransformed validation MAE values, the right plot a histogram of square root transformed validation MAE values.

In the second step of this analysis, a regression tree model is fitted to the experiment result data. As independent variables, the parameters `FSUB`, `FSUB`, `FDIV`, `FSIN`, `FCOS`, `FTAN`, `FEXP`, `FLOG`, and `FSQRT` are used, while the square root of the validation MAE of the best solution found is used a dependent variable. The model fit is performed using Therneau and Atkinson [1997]'s R package rpart with a maximum tree depth of 2.[16]

[16] Therneau, T. M. and Atkinson, E. J. (1997). An introduction to recursive partitioning using the RPART routines. Technical Report 61, Department of Health Science Research, Mayo Clinic, Rochester NY

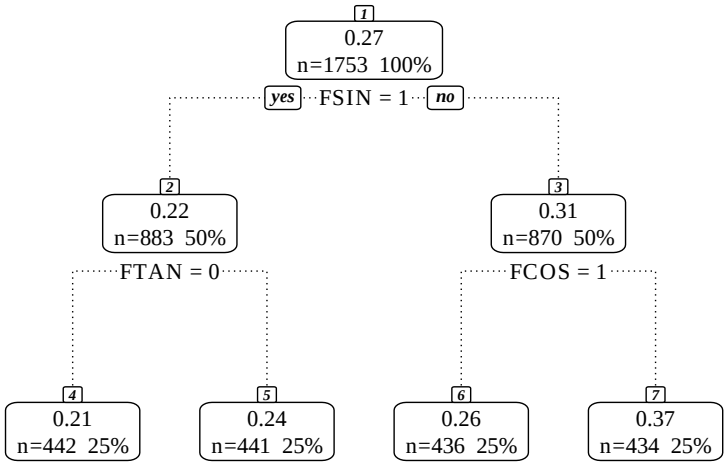


Figure 5.6: GP function set selection decision tree model for the Salustowicz 1D problem class.

Figure 5.6 visualizes the resulting decision tree. This model can be used to predict the expected validation MAE of the best solution found depending on the composition of the GP function set. The root note represents all available 1753 validation MAE values. As 2048 experiments were conducted, 295 (14%) experiments failed to produce GP solutions defined on validation data, e.g. due to numeric instability or invalid operations such as division by zero.

The decision tree model can be interpreted as follows. If sin is included in the GP function set, the expected validation MAE of the best solution found for the Salustowicz 1D problem class is expected to be 0.22. There are 883 observations supporting this prediction. Additionally, if tan is not included in the function set, the expected MAE drops to 0.21 (442 observations). If, on the other hand, tan is included in the function set, the expected MAE raises to 0.24 (441 observations). In the case that sin is not included in the function set but cos is, the expected MAE raises to 0.26 (436 observations). If both sin and cos are excluded, the expected MAE raises still further to 0.37 (434 observations). In summary, to solve problems in the Salustowicz 1D class effectively with GP, the sine or cosine function should be included in the function set, and the tangent should be excluded. While this result matches intuition, it shows that it is important to gain intuition on the input data used for symbolic regression with GP and to choose the GP function set accordingly.

Statistical Hypothesis 6 (H-6) Based on the interpretation of the regression tree model given in the last paragraph, the hypothesis that there exists a proper subset of $\hat{F} \subset F$ that significantly improves GMOGP-FCA system performance on the Salustowicz 1D problem class must not be rejected. All subsets containing the function sin and not containing the function tan are expected to yield significantly better performance than F (which contains both sin and tan).

5.7 GP System Parameter Tuning

The final series of experiments is concerned with tuning the most influential parameters of the RGP system for best performance on a given problem class. This approach to GP parameter tuning is introduced by means of two example problem classes and can easily be applied to any problem class, leading to near-optimal GP system parameter settings for this class. Therefore, the main application area of this approach is tuning a GP system for use in a certain application domain. It is expected that the need for application specific GP systems will rise as GP technology becomes more prevalent in the future.

5.7.1 Experiment Setup

In the following series of experiments, the same basic setup is used as in the previous series, i.e., a compute time budget of four hours per GP run, except for the following modifications. In addition to the Salustowicz 1D test function (P5), Air Passengers 1D (P4) is used as a second test function. The Kriging test function generator was set to a difficulty range of $[0.8, 1.2]$ for both test functions. The GP function set was fixed to $\{+, -, \times, \div, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$

for test function P4 and to $\{+, -, \times, \div, \sin, \cos, \exp, \log, \sqrt{x}\}$ for test function P5, based on results of the previous section on GP function set selection. For both test functions, the error measure was fixed to SMSE.

5.7.2 *Statistical Hypotheses*

Experiments performed in this section are mainly concerned with testing the scientific claims that the parameter settings of modern GP systems can be effectively tuned (claim C-2) and that tuned GP system parameter settings are robust within a problem class (claim C-3). As two distinct problem classes are examined in this section, first insights into possible differences between tuned parameter settings for different classes are also possible.

Statistical Hypothesis 7 (H-7) *There exist problem classes where* GMOGP-FCA *system performance as measured by validation* MAE *of the best solution found can be measurably improved by* SPO*.*

This hypothesis states that there are problem classes where GP system parameters can be effectively tuned and that the performance improvement gained by SPO is robust to changes of the concrete problem instance within a problem class.

Statistical Hypothesis 8 (H-8) *There exist pairs of problem classes with measurable differences in their* SPO*-tuned* GMOGP-FCA *system parameter settings.*

According to this hypothesis, no single set of default GP system parameter settings will be appropriate for every problem class, i.e., for best results, SPO has to be performed separately for each GP application domain. This is in accordance with findings from other authors, e.g., De Jong [2007] beliefs that EAs pre-tuned with default parameter values for particular problem classes will continue to provide better performance than EAs that attempt to dynamically adapt too many control parameters.[17]

[17] De Jong, K. A. (2007). Parameter setting in EAs: a 30 year perspective. In Lobo, F. G. et al., editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 1–18. Springer, Berlin

5.7.3 *Experiment Design*

The GP system parameter set tuned by SPO is summarized in Table 5.14. As mentioned above, the GP function set was set to $\{+, -, \times, \div, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$ for P4 and to the same set without tan for P5. The error measure was set to SMSE in all GP runs. Other GMOGP-FCA parameters not mentioned in this table were set to default values.

The parameter space selected for SPO consists of five continuous parameters and one integer parameter. For the tuning experiments for both test functions P4 and P5, SPOT was configured as shown in Table 5.13. The previous Chapter (4) described SPOT in more detail. To recapitulate and put the settings shown in Table 5.13 into context, SPOT starts with an initial Latin Hypercube design

	Variable (Symbol)	*Domain*
Population Size	`mu` (μ)	$\{8, \ldots, 256\}$
Children per Generation (relative)	`lambdaRel` ($\lambda_{\mu\,\mathrm{rel}}$)	$[0, 1]$
New Individuals per Generation (relative)	`nuRel` ($\nu_{\mu\,\mathrm{rel}}$)	$[0, 1]$
Crossover Probability	`crossoverP` ($p_{\mathrm{crossover}}$)	$[0, 1]$
Constant Mutation Weight	`constantMutationPw` (p_{mconst})	$[0, 1]$
Parent Selection Probability	`parentSelectionP` (p_{psel})	$[0, 1]$

Table 5.12: Parameters of the RGP system with GMOGP search heuristic tuned by SPO.

of size three (Initial Design), i.e., with 32 configurations of the GP system. Each configuration is repeated two times (Initial Repeats) for two test function instances each, resulting in 128 GP runs being executed in the initial SPO step. The results of the initial step are then employed to train a Random Forest model (Meta Model). A sequential design of 10000 GP system configurations (Sequential Design Size) is evaluated on this Random Forest model. The 32 GP system configurations (Sequential New Design Size) found based on the predictions of this meta-model are then optimized using the Limited-Memory Broyden–Fletcher–Goldfarb–Shannon Algorithm with Box-Constraints (Meta Model Optimizer, L-BFGS-B). These 32 configurations are then evaluated on the GP system, where each distinct configuration is evaluated up to four times (Sequential Maximum Repeats). In the sequential phase of SPO, the model fitting, prediction, and evaluation steps are repeated until a preset budget of 1024 GP system evaluations (Sequential Evaluations) is reached. Finally, the best GP configuration found is reported.

Parameter	*Setting*
Initial Design	Latin Hypercube (Size 32)
Initial Repeats	2
Meta Model	Random Forest
Meta Model Optimizer	L-BFGS-B
Sequential Evaluations	1024
Sequential Maximum Repeats	4
Sequential Design Size	10000
Sequential New Design Size	32

Table 5.13: SPOT configuration for GP parameter tuning.

Using the SPOT configuration described in Table 5.13, for each of the two test function classes, 128 + 1024 = 1152 GP runs with a compute time budget of four hours each are executed on a compute cluster. With this configuration, SPOT divided its budget into four sequential steps with 256 GP runs each.

5.7.4 *Experiment Results*

In the two test function classes studied, tuning GMOGP-FCA parameter settings with SPO resulted in clear performance improvements. On the Air Passengers 1D test function class (P4), the median validation MAE of 16.7881 (SD 9.8069) based on default pa-

rameters was improved to 13.2803 (SD 2.6875), an improvement of 26.41 %. On the Salustowicz 1D test function class (P5), the superiority of the tuned GP system parameters was even more pronounced with a tuned median validation MAE of 0.0169 (SD 0.0085) compared to a default median validation MAE of 0.0304 (SD 0.085), an improvement of 79.88 %. Note that the standard deviation of the tuned results was also about an order of magnitude smaller.

Parameter	*Default*	*Tuned for P4*	*Tuned for P5*
μ	100	198	75
$\lambda_{\mu\,\mathrm{rel}}$	1	0.51	0.73
$\nu_{\mu\,\mathrm{rel}}$	0.5	0.57	0.55
$p_{\mathrm{crossover}}$	0.5	0.84	0.65
p_{mconst}	0	0.72	0.6
p_{psel}	1	0.06	0.54

Table 5.14: SPO-tuned parameter settings for the RGP system with GMOGP-FCA search heuristic, Air Passengers 1D (P4) and Salustowicz 1D (P5) test function classes.

TABLE 5.14 compares default GMOGP-FCA system parameter settings to settings tuned for test function classes P4 and P5, respectively. There are some notable differences in the tuned parameter settings. The Air Passengers 1D test function class (P4) requires larger GP population sizes (μ) compared to the Salustowicz 1D test function class (P5). P4 requires a both relatively and absolutely smaller number of children per generation ($\lambda_{\mu\,\mathrm{rel}}$) compared to P5. The relative number of newly initialized individuals per generation ($\nu_{\mu\,\mathrm{rel}}$) is about the same for both P4 and P5. The differences in crossover and constant mutation probabilities are measurable but minor. Interestingly, a higher probability of performing parent selection (p_{psel}) seems to harm P4 but benefit P5. In general, test function class P4 seems to benefit from a more exploratory GP system configuration, while P5 seems to benefit from a configuration that exploits existing solutions to a higher degree. Also, results on both test function classes seem to benefit from an enabled constant mutation operator.

To investigate the effect of the relative number of children ($\lambda_{\mu\,\mathrm{rel}}$) and the relative number of newly initialized individuals ($\nu_{\mu\,\mathrm{rel}}$) on GMOGP-FCA system performance, contour plots were generated for both test function classes P4 and P5 based on meta-model fits on the SPO experiment result data. These plots are reproduced in Figure 5.7. Here, support vector machine meta-models were used that are slower to fit than random forest models, but offer slightly better fidelity and smoother response values that are easier to plot.

Regarding test function P4, the global optimum of $\lambda_{\mu\,\mathrm{rel}} = 0.51$ and $\nu_{\mu\,\mathrm{rel}} = 0.57$ is clearly visible as the dark area at the center of the left contour plot. A pronounced local optimum seems to be located at the maximum value for $\lambda_{\mu\,\mathrm{rel}}$ and a lower value for $\nu_{\mu\,\mathrm{rel}}$, as visible in the region with $\lambda_{\mu\,\mathrm{rel}} > 0.75$ and $\nu_{\mu\,\mathrm{rel}} < 0.5$ of the left contour plot of Figure 5.7.

Regarding test function P5, the global optimum of $\lambda_{\mu\,\mathrm{rel}} = 0.73$

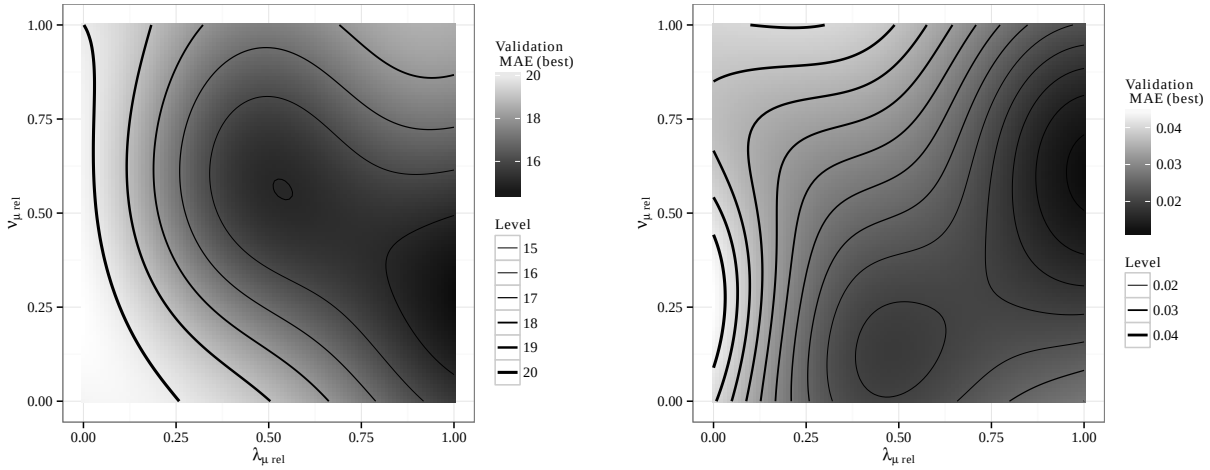


Figure 5.7: Contour plots of the effect of parameters $\lambda_{\mu\,\mathrm{rel}}$ (x-axis) and $\nu_{\mu\,\mathrm{rel}}$ (y-axis) on GMOGP-FCA system performance (gray level). Other parameters are fixed at their respective optima found by SPO. Results for test function class P4 are shown on the left, while results for test function class P5 are shown on the right.

and $\nu_{\mu\,\mathrm{rel}} = 0.55$ is visible as the dark area at the region with $\lambda_{\mu\,\mathrm{rel}} > 0.75$ of the right contour plot. A less pronounced local optimum at a medium value for $\lambda_{\mu\,\mathrm{rel}}$ and a low value for $\nu_{\mu\,\mathrm{rel}}$ is visible in the region with $\nu_{\mu\,\mathrm{rel}} < 0.25$ of the right contour plot. These local optima illustrate the multi-modal nature of GP system parameter optimization.

Statistical Hypothesis 7 (H-7) The hypothesis that there exist problem classes where GMOGP-FCA system performance as measured by validation MAE of the best solution found can be measurably improved by SPO cannot be rejected. At least on test function classes P4 and P5, performance benefits of tuned GP system parameters are clearly apparent.

Statistical Hypothesis 8 (H-8) Also, the hypothesis that there exist pairs of problem classes with measurable differences in their SPO-tuned GMOGP-FCA system parameter settings cannot be rejected based on the experimental results described above. The tuned GMOGP-FCA parameter settings for test function classes P4 and P5 are clearly different, as shown in Table 5.14. Further research would be necessary to show if there are at least local optima in GMOGP-FCA system parameter space shared by many practical problem classes. These local optima would be good candidates for reasonable GMOGP-FCA default system parameter settings.

5.8 Conclusions

Based on the framework of principled empirical research provided by SPO, this chapter provided an extensive study of the effect of parameter settings of a modern, i.e., multi-objective, GP system used for symbolic regression. By applying the methods introduced in the previous chapter (4) on SPO, it was possible to identify the most important GP parameters, select near-optimal GP function sets for

given problem classes, and set near-optimal GP algorithm parameters. The methods and procedures used are general enough to be applied to other GP system components, furthering principled and reproducible research. The combined use of statistical modeling and visualization enabled detailed insights into the effects of different parameter settings on the expected multi-objective performance of the GP result population.

Results of this chapter are useful to guide GP research as well as practical application. It is now possible to test the plausibility of the scientific claims formulated in Section 5.2 and to formulate best practices for GP system optimization in real-world applications.

5.8.1 Scientific Claims

By formulating and testing several statistical hypothesis based on the scientific claims introduced at the beginning of this chapter, it is now possible to examine the validity of these claims in more detail. Due to their general formulation and in contrast to the statistical hypotheses presented in this work, no definitive answer regarding the validity of these claims can be expected, only more or less strong hints based on empirical results.

Scientific Claim 1 (C-1) The claim that complex problems require complex algorithms seems to withstand empirical scrutiny. The results of testing hypotheses H-1 and H-2 clearly indicate that GP system performance is reversely correlated to problem difficulty, and that more modern and complex GP algorithms significantly and consistently beat simpler GP algorithms in performance. Based on these results, it is likely that tailoring GP search heuristics further to the expected search space structure, for example by adding algebraic domain knowledge for symbolic regression, will increase GP algorithm performance even further.

Scientific Claim 2 (C-2) The claim that parameter settings of modern GP systems can be effectively tuned appears to be valid. By testing hypotheses H-3 and H-4, it was shown that for at least the GP system used in this work, there exist parameters that have a significant effect on system performance as measured by validation MAE of the best solution found, and that there are significant differences in the relative effects of the parameters studied. Furthermore, the result of testing hypothesis H-5 indicates that the choice of the GP function set, a GP parameter stetting, also has an significant effect on GP performance. Experimental results as presented in Section 5.7.4 demonstrate that GP system parameter tuning is effective in practice.

Scientific Claim 3 (C-3) The claim that tuned GP system parameter settings are robust within a problem class appears to hold up to first empirical study. The results of testing hypotheses H-7 and H-8

clearly show that there are problem classes where GP performance as measured by validation MAE of the best solution found can be measurably improved by SPO and that there are pairs of problem classes with measurable differences in their SPO-tuned parameter settings. These results at least hint at the robustness of tuned parameters within a problem class, yet further experimentation with other problem classes would be necessary to secure this claim.

5.8.2 Best Practices for GP System Optimization

Based on the process described in this chapter, a first attempt at a set of best practices for GP system parameter tuning can be proposed. GP system parameter tuning, including automatic GP function set selection, as described in this chapter, can significantly improve GP performance for a given problem class. As tens to hundreds of individual GP runs are necessary for the tuning process, the compute time cost can be substantial, though.

Whether the expected performance improvement relative to default parameters is worth the cost has to be decided on a case-by-case basis. Generally speaking, if GP is to be repeatedly applied within the same problem class, for example to update existing models to changing conditions, the compute time spent tuning GP system parameters via SPO will be more than redeemed trough savings due to better GP algorithm performance.

In the case only a limited compute time budget is available for GP system parameter tuning, only the parameters with highest expected impact should be tuned. As seen in the analysis of hypothesis H-4, these are the function set (i_{funcset}), the error measure (i_{error}), and the population size (μ). Concentrating the tuning effort on these parameters while keeping all other parameters at default values allows the reduction of the total experiment budget available to SPOT, reducing compute time considerably.

6
Case Studies

Artificial test functions are an important tool for benchmarking GP systems and for testing hypotheses about the benefits or drawbacks of GP system components and their parameter settings. In the previous chapters of this work, these questions were studied in detail within the framework of the modular GP system RGP.

To put these results into context, comparisons to other state-of-the-art GP systems are necessary. Also, practitioners will be interested in comparisons with alternatives to GP, such as methods from machine learning or classical statistics. Ideally, these comparisons should be performed using well-known test problems as well as real-world examples, the latter providing a guide for using GP in a real-world setting. The current chapter addresses these requirements by providing comparisons with other state-of-the-art GP systems, both commercial and open-source, and by providing two extensive case studies of applying RGP successfully to real-world applications.

The remainder of this chapter is structured as follows: Artificial test problems used to compare RGP to other state-of-the art GP systems will be introduced and applied in the following section. Next, the design and organization of the case studies based on real-world applications will be described. This section will also shortly introduce alternative regression techniques used in later comparisons. After that, the two real-world case studies will be described in detail. The first is a meta model generation task for cyclone dust separator optimization. The second is a control model generation task for steel roll trains. This chapter closes with a summary and conclusions.

6.1 Artificial Test Problems

This section uses three artificial test problems to assess the performance of RGP configured with the GMOGP-FCA search heuristic as described in Section 2.9.3. All of these seem to be common in the GP literature or, in the case of the Air Passengers 1D test problem, in the literature about time series forecasting. All artificial test functions introduced in this section can be classified as symbolic regression problems.[1]

[1] Data and R source code for all experiments of this section are available at https://rsymbolic.org/projects/ofdiss/repository/revisions/master/show/experiments/rgpValidation.

Artificial test problems may not reflect the typical complexity of real-world applications, but are nonetheless useful for multiple reasons. First, they are simple enough to be easily understood without extensive domain knowledge. Second, the ground truth, i.e., the true function or data is known, enabling optimization algorithm analysis techniques that require knowledge of the true global optimum, such as fitness distance correlation analysis. Third, due to their prevalence, results based on these test functions can be compared to exiting results from the literature.

6.1.1 *Air Passengers 1D*

The Air Passengers 1D test function has been introduced as a base function for the Kriging-based test function generator P4 (see Section 4.3.2). To enable comparisons with existing literature, here it is used directly as a fixed test function, though. As mentioned, this test function is based on the "air passengers" data set first introduced by Box et al. [2008].[2] The data consists of monthly totals of international airline passengers, from 1949 to 1960. The independent variable ranges over the number of months since January 1949, the dependent variable being the number of airline passengers in thousands. Strictly speaking, Air Passengers 1D is a real-world data set, though a exceptionally small one, which makes it well suitable as a test problem.

[2] Box, G., Jenkins, G., and Reinsel, G. (2008). *Time-Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, Hoboken, NJ

The training interval is defined as $[1, 108]$, the validation interval $[109, 144]$, and the number of fitness cases is fixed to $N_{\text{Air Passengers 1D}} := 144$. Unless stated otherwise, the GP function set used is with this test function is $\{+, -, *, /, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$.

6.1.2 *Kotanchek 2D*

The Kotanchek 2D test function has been first introduced by Smits and Kotanchek [2004] and found broad adoption in the GP community.[3] It is an analytical function modelled after real-world problems in the domain of engineering optimization. Variants of this test function can be generated by changing constants.

[3] Smits, G. and Kotanchek, M. (2004). Pareto-front exploitation in symbolic regression. In O'Reilly, U. et al., editors, *Genetic Programming Theory and Practice II*, Genetic and Evolutionary Computation Series, pages 283–299. Springer, New York

$$f_{\text{Kotanchek 2D}}(x_1, x_2) := \frac{e^{-(x_1-1)^2}}{1.2 + (x_2 - 2.5)^2}$$

The training and validation intervals of the Kotanchek 2D test function are defined as $[(0, 0), (4, 4)]$, where a random sample of 50% of the data is used for training, while the remainder is used for validation. The total number of fitness cases is fixed to $N_{\text{Kotanchek 2D}} := 500$. Unless stated otherwise, the GP function set used is $\{+, -, *, /, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$.

6.1.3 *Salustowicz 1D*

As with the Air Passengers 1D test function, this test function has been introduced as a base function for the Kriging-based test function generator P5 (see Section 4.3.2). Here, this test function is used

directly, to enable comparisons with existing literature. As said, the Salustowicz function was first introduced by Salustowicz and Schmidhuber [1997]. It has been designed to be simple to read yet challenging for GP.[4]

[4] Salustowicz, R. and Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141

$$f_{\text{Salustowicz 1D}}(x_1) := e^{-x} x^3 \cos(x) \sin(x) [\sin^2(x) \cos(x) - 1]$$

The training and validation interval of the Salustowicz 1D test function is $[0, 10]$, where a random sample of 50% of the data is used for training, while the remainder is used for validation. The total number of fitness cases is fixed to $N_{\text{Salustowicz 1D}} := 100$. Unless stated otherwise, the GP function set used is $\{+, -, *, /, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$.

6.1.4 *Experiment Setup*

Goal of this experimental study is to put the performance of the RGP system, configured with the GMOGP-FCA search heuristic, into context by comparing it with the performance of other commercial and open source GP systems on three well-known artificial test problems in symbolic regression. Besides RGP, the open source GP system ECJ and the commercial GP system Eureqa are included in this comparison.

As mentioned, ECJ appears to be the most popular open source GP system today, based on number of citations. For that reason alone, it should be included in the set of systems to be compared. Due to its simple to use graphical user interface, generally good performance, and publicity generated by an article in Science [5], Eureqa seems to be the most widespread commercial GP system for symbolic regression today. It is therefore included in the comparison. Section 3.1 contains a more detailed description of these systems, as well as a list of features for each system.

[5] Schmidt, M. D. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85

General Settings For each GP system configuration, ten independent runs with different random seeds have been performed. As stopping criterion, a fixed number of 10,241,024[6] candidate solution evaluations has been chosen. This choice is warranted by the fact that the systems under comparison use widely differing implementation technologies and levels of optimization, and that this study is motivated by comparing the underlying GP algorithms, instead of implementation technologies. On the other hand, an indication of the required wall clock runtime for each system would of course be valuable for practitioners. Unfortunately, no exact time measurements were possible for technical reasons. As an indication, both RGP and ECJ required a runtime of about two hours for performing 10,241,024 candidate solution evaluations. Eureqa required a runtime of about 30 minutes for the same number of evaluations. Of the systems compared, Eureqa appears to offer the most highly optimized implementation of a multi-objective GP search strategy. Note that solution quality does not scale linearly with number of evaluations, though.

[6] This number is motivated by informal calibration runs with the ECJ system, which uses a population size of 1024 by default. These runs resulted in a recommended runtime of 10001 generations.

RGP Version 0.4-0 of RGP, the current version as of January 2014, has been used. As parameter optimization was not part of this comparison, RGP was configured with the GMOGP-FCA search heuristic and set to default parameter values, i.e., a population size of 100 individuals, 50 children and 50 newly initialized individuals per generation, random initialization, selection based on fitness, age, and complexity, and the GP function set $\{+, -, \times, \div, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$. See Section 2.9.2 for details. To obtain reproducible results, the random number generator seed has been set to 1 to 10 for run number 1 to 10, respectively.

ECJ Version 21 of ECJ, the current version as of January 2014, has been used in this study. Much like RGP, ECJ is a framework for constructing evolutionary algorithms, not a system with a fixed set of features. It comes with an extensive example implementing GP for symbolic regression, though. The default settings of this example have been used in this comparison, i.e., a population size of 1024 individuals, Koza-style fitness (see Koza [1992])[7], ramped half-and-half initialization with a maximum tree depth of 9, tournament selection with tournament size of 7, crossover probability of 0.9, and mutation via subtree replacement with newly initialized trees of maximum depth 5. Note that, as of this writing, ECJ does not support multi-objective selection in its example implementation of GP, therefore single-objective selection has been used. Depending on the problem dimensionality, ECJ supports different fixed GP function sets. As no clear indication is given which function set to use for a given problem class, all applicable function sets have been evaluated in this study. For one dimensional data (test functions Air Passengers 1D and Salustowicz 1D), these function sets are Koza1 := $\{+, -, *, /, \sin, \cos, \exp, \log\}$, Keijzer1 := $\{+, *, -x, 1/x, \sqrt{x}\}$, and VladislavlevaC1 := $\{+, -, *, /, x^2, \exp, -\exp, \sin, \cos\}$. For two dimensional data (test function Kotanchek 2D), the applicable function sets are Koza2, Keijzer2, VladislavlevaC2, VladislavlevaA2 := $\{+, -, *, /, x^2\}$, and VladislavlevaB2 := $\{+, -, *, /, x^2, \exp, -\exp\}$. Table 6.1 summarizes this information. In ECJ, input variables and ephemeral constants are considered part of the function set. Besides the additional input variables, the function sets Koza2, Keijzer2, and VladislavlevaC2, are the same as Koza1, Keijzer1, and VladislavlevaC1, respectively. In ECJ, Koza-style function sets do not include any constants, while Keijzer-style and Vladislavleva-style function sets use slightly different types of ephemeral constants. See the documentation of ECJ version 21 for details.[8] As with RGP, to generate reproducible results, the random number generator seed has been set to 1 to 10 for run number 1 to 10, respectively.

Eureqa Version 8.16 of Eureqa, the most current version as of January 2014, has been used. Eureqa counts the evaluation of individ-

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

[8] Luke, S. (2013). The ECJ owner's manual – A user manual for the ECJ evolutionary computation library. 21th edition, http://cs.gmu.edu/ eclab/projects/ecj/docs/manual/manual.pdf (retrieved 20.02.2015)

	$+$	$-$	$-x$	$*$	$/$	$1/x$	$\sqrt{x}$	x^2	exp	$-$ exp	log	sin	cos
Koza1	✓	✓	–	✓	✓	–	–	–	✓	–	✓	✓	✓
Keijzer1	✓	–	✓	✓	–	✓	✓	–	–	–	–	–	–
VladislavlevaC1	✓	✓	–	✓	✓	–	–	✓	✓	✓	–	✓	✓
Koza2	✓	✓	–	✓	✓	–	–	–	✓	–	✓	✓	✓
Keijzer2	✓	–	✓	✓	–	✓	✓	–	–	–	–	–	–
VladislavlevaA2	✓	✓	–	✓	✓	–	–	✓	–	–	–	–	–
VladislavlevaB2	✓	✓	–	✓	✓	–	–	✓	✓	✓	–	–	–
VladislavlevaC2	✓	✓	–	✓	✓	–	–	✓	✓	✓	–	✓	✓

Table 6.1: Overview of the ECJ function sets used in this work.

ual fitness cases, which has been taken into account when defining the stopping criteria for each test function. The default Eureqa function set $\{+, -, *, /, \sin, \cos\}$ has been used for all test functions. Ephemeral constants were enabled per default. All other parameters were also set to their default values, i.e., MAE and solution complexity as selection criteria. All other parameters of the GP algorithm were set automatically by the system, with no means of manual intervention. Eureqa uses a time-based random seed and has no option to set a user-defined seed, unfortunately. As a consequence, results are not reproducible. However, result variance indicates that results of repeated runs should be comparable.

6.1.5 *Results*

This subsection presents experimental results for each test function and GP system in the form of box plots and tables. A general discussion of the results will follow in the next subsection.

Air Passengers 1D Figure 6.1 and Table 6.2 summarize the validation MAE values of the best models found after 10,241,024 fitness, i.e., candidate solution, evaluations based on the Air Passengers 1D test function. Best values are marked with bold font in the table. Note that the coefficient of determination (R^2) which is often provided as a measure of model accuracy is only defined for linear regression analysis, and is therefore omitted in the presentation of these results.

	RGP	*Eureqa*	*ECJ.Koza1*	*ECJ.Keijzer1*	*ECJ.Vlad.C1*
Median	43.5089	**27.7093**	52.4425	52.7964	59.2843
SD	10.1922	5.2818	**0.4625**	0.7369	7.3665
Rank	2	**1**	3	4	5

Table 6.2: Validation MAE of the best models found after 10,241,024 fitness evaluations for the Air Passengers 1D test function. Best values are marked with bold font.

Ranked by validation MAE based on the Air Passengers 1D test problem, Eureqa leads with RGP and ECJ (Koza1 function set) taking the second and third place, respectively. Performance differences across different GP systems appear to be significant, while the performance difference between the Koza1 and Keijzer1 function sets for ECJ seems to be negligible. This is interesting because mod-

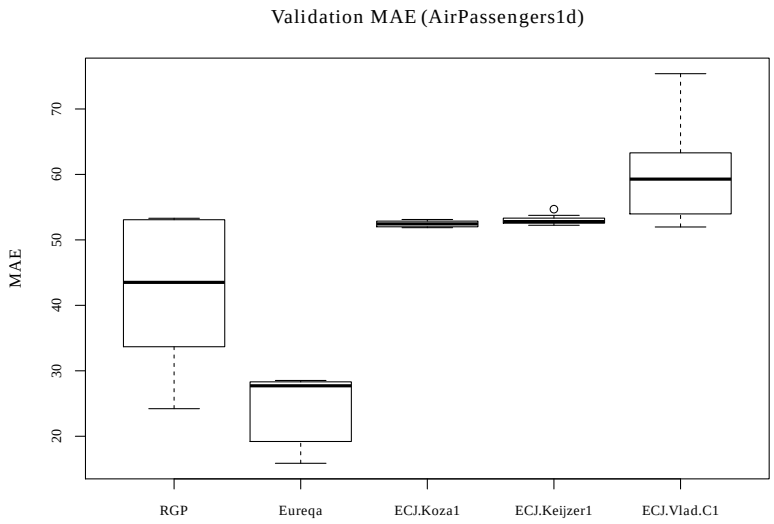


Figure 6.1: Validation MAE of the best models found after 10,241,024 fitness evaluations for the Air Passengers 1D test function.

els based on the Koza1 function set do not contain any constants. The low variance in the results based on these two function sets can be explained by inspecting the generated models. In nearly all runs, only approximations of simple linear models without periodic components were found. The genotypic representation of these models was nonetheless quite large and approaching the maximum tree depth, indicating bloat, which is a typical problem of GP with single-objective selection.

Neither result produces a perfect or even very good fit on validation data. Because Air Passengers 1D tests the extrapolation ability of models fitted on very few data, it is questionable whether enough information is present in the training data to derive near-perfect models. The results presented in Section 5.7.4 hint at space for improvement, though. Tuning GMOGP-FCA parameter settings with SPO resulted in a median validation MAE of 13.2803 (SD 2.6875) for this test problem, which is a very good fit, based on a fixed runtime budget of four hours.

Kotanchek 2D Figure 6.2 and Table 6.3 summarize the validation MAE values of the best models found after 10,241,024 fitness, i.e., candidate solution, evaluations based on the Kotanchek 2D test function. As before, best values are marked with bold font in the table.

	RGP	*Eureqa*	*ECJ.Koza2*	*ECJ.Keijzer2*	*ECJ.Vlad.A2*	*ECJ.Vlad.B2*	*ECJ.Vlad.C2*
Median	0.0158	**0.0084**	0.0153	0.0196	0.0238	0.0246	0.0149
SD	0.0080	**0.0027**	0.0057	0.0039	0.0100	0.0090	0.0088
Rank	4	**1**	3	5	6	7	2

Table 6.3: Validation MAE of the best individuals found after 10,241,024 fitness evaluations for the Kotanchek 2D test function. Best values are marked with bold font.

Ranking the different GP systems by validation MAE based on

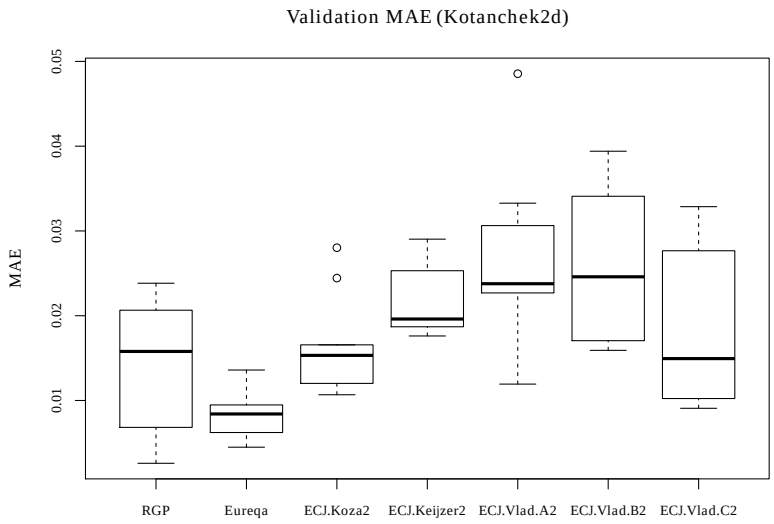


Figure 6.2: Validation MAE of the best individuals found after 10,241,024 fitness evaluations for the Kotanchek 2D test function.

the Kotanchek 2D test problem results in the following picture. Eureqa is able to generate results with the lowest median validation MAE of 0.0084, with ECJ being second (VladislavlevaC2 function set, 0.0149) and third (Koza2 function set, 0.0153), and RGP being close fourth with 0.0158. Overall, performance differences are less pronounced than with the Air Passengers 1D test function. None of the models found produces a perfect fit on validation data. The best model found in all ten runs has a validation MAE of 0.0026 and was discovered by RGP.

Salustowicz 1D Figure 6.3 and Table 6.4 summarize the validation MAE values of the best individuals found after 10,241,024 fitness, i.e., candidate solution, evaluations based on the Salustowicz 1D test function. As before, best values are marked with bold font in the table.

	RGP	*Eureqa*	*ECJ.Koza1*	*ECJ.Keijzer1*	*ECJ.Vlad.C1*
Median	**0.0246**	0.0317	0.0491	0.1237	0.0851
SD	0.0117	**0.0074**	0.0517	**0.0074**	0.0434
Rank	**1**	2	3	5	4

Table 6.4: Validation MAE of the best individuals found after 10,241,024 fitness evaluations for the Salustowicz 1D test function. Best values are marked with bold font.

Ranked by median validation MAE based on the Salustowicz 1D test function, the RGP system takes the first place, followed by Eureqa, and by ECJ (Koza1 function set) as a distant third. Nonetheless, ECJ was the only system that found a perfect solution, i.e., a solution with validation MAE of zero, during one of its ten runs. Performance differences are significant in the results created for this test function. Even more striking is the high variance of the results created by ECJ when configured with either the Koza1 or VladislavlevaC1 function set. With both function sets, ECJ is able to

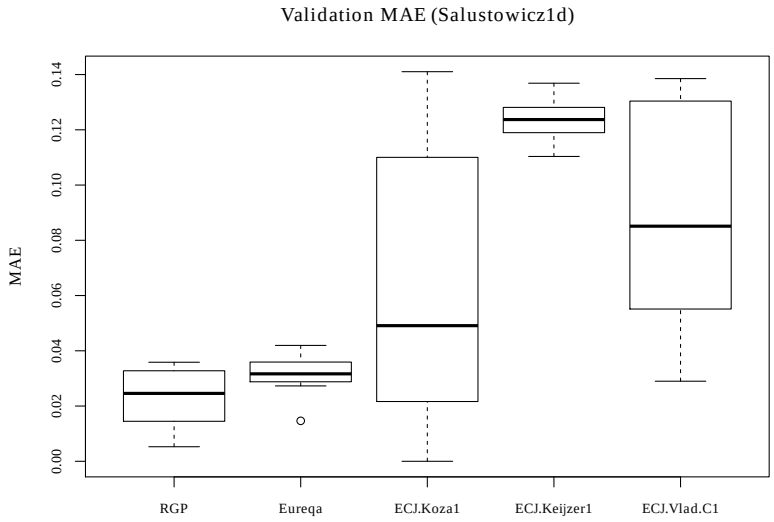


Figure 6.3: Validation MAE of the best individuals found after 10,241,024 fitness evaluations for the Salustowicz 1D test function.

discover good models in some runs, while getting stuck in local optima in other. Further analysis shows a loss of population diversity in the latter case, hinting at measures for diversity preservation as a possible solution.

The models found by RGP and Eureqa for this test function are generally of very high quality. Even better results are possible by GP system parameter tuning or by increasing the compute time budget, as demonstrated in Section 5.7.4.

6.1.6 Discussion

There are several insights to be gained from the results described in the previous section. First of all, while designing a comparison study of different GP systems is a challenging endeavour, designing a comparison study that results in a system ranking valid for all users and applications seems to be entirely impossible. This is due to multiple factors. First, relative performance of GP systems depends on the compute time budget available. Second, relative GP system performance is dependent on the application problem, as evident in the results presented. Third, and perhaps trivially so, GP system performance is dependent on the optimization criterion or criteria chosen. Practitioners are strongly advised to compare GP systems in their specific application domain, tuning the GP system parameter settings via a principled approach like SPO, to achieve meaningful results.

This comparison is based on three well-known artificial symbolic regression test problems, and therefore lacks a specific application domain. Parameters were not tuned, but set to system default values to keep the compute time budget requirements of this study manageable. This included highly significant parameters, such as

the GP function set, meaning that different GP systems operated on different GP search spaces. While these can be seen as weaknesses of this study, some general trends are nonetheless evident from the results. Overall, multi-objective search heuristics like the ones implemented in Eureqa and RGP offer a performance benefit over single-objective strategies (see Section 6.1.5). As a positive side effect, the solutions found by Eureqa and RGP do not suffer from bloat to the same degree as solutions generated by ECJ. As a second result, based on the test functions studied, the performance of the RGP system configured with the GMOGP-FCA search heuristic, as measured by median validation MAE of the best solution found in each run, is at least competitive with both ECJ and Eureqa.

6.2 *Real-World Case Study Design and Organization*

Table 6.5 summarizes the real-world case studies discussed in this section. Each case study follows the process for model induction with RGP described in Section 3.2. To recapitulate, this process comprises the steps of modeling project setup, problem definition, data acquisition, data preprocessing, data partitioning, classical modeling, genetic programming, model validation, and model deployment.

Case Study	*Problem Class*	*Application Domain*
AppDust	Regression	Meta-Model Generation
AppSteel	Regression	Control Model Generation

Table 6.5: Summary information for the real-world case studies described in this thesis.

In this work, the RGP process step of classical modeling for regression problems has been broadly extended to cover not only linear models, but also Kriging, multivariate adaptive regression splines, principal component regression, random forests, and support vector machines. In a typical industrial setting, only a subset of these alternative regression modeling techniques would be evaluated, due to time and cost constraints. Here, all of these techniques have been evaluated in both case studies, to put results generated by RGP into context. In the following subsections, these techniques are introduced briefly. Please follow the references given in the individual subsections for more details.

Generally, a regression technique models the relationship between a dependent (or response) variable $\mathbf{y}$ and n independent (or explanatory) variables $\mathbf{x_i}$, where $i \in \{1, \ldots, n\}$. If multiple dependent variables are to be modeled, techniques for multivariate regression can be used, which are not in the focus of this work, though.

6.2.1 *Linear Models*

A linear model (LM) assumes that the relationship between dependent variable and independent variables is linear:

$$\mathbf{y} = \beta_1 \mathbf{x_1} + \cdots + \beta_n \mathbf{x_n} + \boldsymbol{\epsilon} \tag{6.1}$$

Here, $\mathbf{y}$ and $\mathbf{x_i}$ are real-valued data vectors. Fitting a linear model means estimating the parameters β_i. The most widely used method in this regard is ordinary least-squares estimation. If the distribution of the error terms $\boldsymbol{\epsilon}$ is normal, least-squares estimation is equivalent to maximum-likelihood estimation.

Linear models are very well understood and have an extensive theoretical foundation, which makes it possible to generate valuable knowledge from these models easily. For example, the widely used technique of ANOVA can be interpreted in the linear regression framework. Results are comparatively easy to analyze and comprehend. Generalized linear models, an extension of the technique, allow the transfer of many methods of linear modeling and analysis to data which is bounded or discrete, i.e., non-linear in a form that is known a priori. There are efficient algorithms for fitting linear models with highly optimized implementations, making them applicable to large datasets.

Naturally, linear models are not able to capture non-linear relationships in a meaningful way. In practice, special care must be taken that all model assumptions are met during data preprocessing and model validation, cf. Figure 6.4. A certain degree of statistical expertise is required to avoid errors and to use the large tool set of linear modeling to its full potential.

The tools built into R were used for all linear modeling conducted in this work. See R's online documentation for the `lm` function for details. Adler [2010] provides a comprehensive introduction to these tools, as well as further literature references.[9]

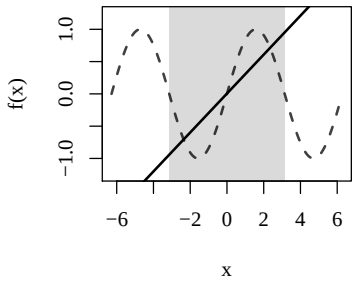


Figure 6.4: Linear model fit of the (strongly non-linear) sine function. Model predictions are shown as a solid black line, the true function is shown as a gray dashed line. The training interval is marked with a gray background.

[9] Adler, J. (2010). *R in a nutshell*. O'Reilly, Sebastopol, CA

6.2.2 *Principal Component Regression*

In Principal Component Regression (PCR), a principal component analysis (PCA) is performed on the independent variables, then a linear model is fitted on the result.[10] The method is comprised of three steps. First, a PCA is conducted on the data matrix formed by the independent variables, yielding their principal components. Next, a linear regression model is fitted on a suitable subset of these component vectors, yielding a vector of estimated model parameters β_i, whose dimension is equal to the number of components in the selected subset. Finally, this vector is transformed back to the scale of the independent variables, changing its dimension to be equal to the number of independent variables.

Selecting only a subset of all possible components in the second step of the modeling process can be seen as a form of regularization, benefiting model robustness and efficiency. There are several techniques of deciding how many components to select. In the PCR

[10] Jolliffe, I. (1982). A note on the use of principal components in regression. *Applied Statistics*, 31(3):300–303

runs performed here, a ten fold cross validation is performed on training data to select the best performing set of components.

In its classical form, PCR is a form of linear modeling and therefore carries the same assumptions. When all components are selected, PCR is equivalent to ordinary least squares regression. Extensions of the method exist that replace the linear model with a kernel machine, e.g. an SVM, to support non-linear regression. A practical advantage of PCR is a greater robustness to the detrimental effects of spurious independent variables when compared to classical linear regression.

PCR modeling has been conducted using the R package `pls`.[11] See R's online documentation on the function `pcr` for details.

[11] Mevik, B., Wehrens, R., and Liland, K. H. (2013). *pls: Partial Least Squares and Principal Component regression*. R package version 2.4-3, http://CRAN.R-project.org/package=pls (retrieved 20.02.2015)

6.2.3 *Multivariate Adaptive Regression Splines*

Multivariate adaptive regression splines (MARS) are a non-parametric regression technique that fits a piecewise linear function to support non-linearity. It can be seen as an extension to linear models, therefore some of the theory and means of analysis of LM also apply to MARS.[12]

[12] Friedman, J. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67

A MARS model is a weighted sum of the output of certain basis functions. A basis function is either a constant, a *hinge function*, or a product of multiple hinge functions for modeling variable interaction. Hinge functions are defined as $h_l(x) := \max(0, c - x)$ or $h_r(x) := \max(0, x - c)$ for a constant c, i.e., they have a linear part discontinuously followed by a constant part (or vice versa, for h_r). A sum of hinge functions and constants comprises a piecewise linear function. MARS models are build by recursive partitioning, much like tree models. See Friedman [1991] for details.

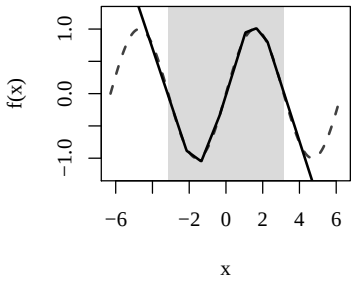


Figure 6.5: MARS model fit of the sine function. The same graphical conventions as in Figure 6.4 have been used.

MARS models are efficient to fit and among the conceptually simpler forms of non-parametric regression techniques. As a blend of ideas from linear regression and decision trees, they are both theoretically and practically well understood. On the other hand, larger MARS models are increasingly hard to understand. If model understandability is not a concern, kernel methods or other non-parametric modeling techniques may give better results. As a trivial example, Figure 6.5 shows a MARS model fit of the sine function. The training data region is marked with a gray background. Note the satisfactory interpolative and unsatisfactory extrapolative performance of the model.

For MARS modeling, the R package `earth` has been used.[13] See the R online documentation on the function `earth` for details.

[13] Milborrow, S. (2014). *earth: Multivariate Adaptive Regression Spline Models*. R package version 3.2-7, http://CRAN.R-project.org/package=earth (retrieved 20.02.2015)

6.2.4 *Random Forests*

Random forests (RF) use an ensemble of decision trees for regression or classification. Bootstrap aggregating (also known as *bagging*) is used both on the training set rows (samples) as well as on the training set columns (features), to counter model overfitting typical to deep regression trees. The algorithm trains an ensemble of

regression trees by repeatedly sampling, with replacement, a random subset of training set rows for training a single tree. Training is performed using recursive partitioning, where each candidate split only sees a random subset of the independent variables. This way, independent variables that are particularly strong predictors of the dependent variables are kept from occurring in every tree, preventing the trees from becoming correlated. For predicting the dependent variable for new data, the mean of the predictions of all trees is used. The optimum number of trees to form the ensemble, typically hundreds to a few thousand, can be selected by monitoring the out-of-bag error.

Random forest models trade the understandability of single decision tree models for better model performance. In many benchmarks and practical applications, they are among the best performing models for both regression and classification tasks. They are comparatively simple and can be efficiently implemented. Figure 6.6 shows a random forest model fit of the sine function. By construction, model extrapolation performance is weak, because the last value at the training interval bounds is returned by the underlying trees. Interpolation performance is very good, though. See Breiman [2001] for more details.[14] R's `randomForest` package has been used for regression modeling with random forests.[15] See the R online documentation on the function `randomForest` for details.

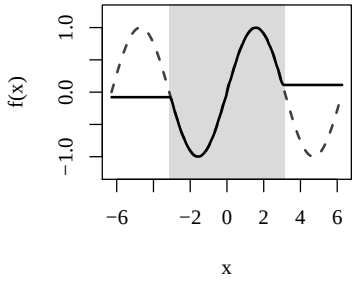


Figure 6.6: Random forest model fit of the sine function. The same graphical conventions as in Figure 6.4 have been used.

[14] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32

[15] Liaw, A. and Wiener, M. (2002). Classification and regression by randomForest. *R News*, 2(3):18–22. R package version 4.6.10, http://CRAN.R-project.org/doc/Rnews/ (retrieved 20.02.2015)

6.2.5 *Support Vector Machines*

Support vector machines (SVM), or support vector networks, are generalized linear classifiers that simultaneously minimize empiric classification error and maximize the geometric margin between classes. They can be seen as an extension of the classical Perceptron artificial neural network that support non-linear classification by means of a suitable transformation function, or kernel. Soft margin SVMs support mislabeled training examples and are the most prevalent SVM variant today, were introduced by Cortes and Vapnik [1995].[16] Originally, the method is only applicable to binary classification, but extensions exists for multi-class classification and regression. The latter, known as support vector regression and introduced by Drucker et al. [1997], is used here.[17]

Benefits of the method include high flexibility and typically high accuracy. Efficient algorithms and optimized implementations exist for both model fitting and prediction. The flexibility of the method can also be seen as a liability, as a certain level of expertise in machine learning is required for selecting a suitable kernel for a given problem and setting kernel and method parameters. Interpreting SVM model fits is difficult and an active area of research. For practical purposes, at least at the time of writing, SVM models should be considered as black-box models. Figure 6.7 shows a support vector regression model fit of the sine function. A radial basis function kernel and epsilon regression has been used for all SVM models

[16] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297

[17] Drucker, H., Kaufman, B. L., Smola, A., and Vapnik, V. (1997). Support vector regression machines. In *Advances in Neural Information Processing Systems*, volume 9, pages 155–161

presented in this work. In the trivial sine example, interpolation performance is good, with gradual decay while moving into the extrapolative region.

In this work, all support vector regression modeling was conducted trough the R package `e1071` based on the `libsvm` library, which is one of the most widely used and feature-rich SVM implementations available.[18] See R's online documentation for the `svm` function for details.

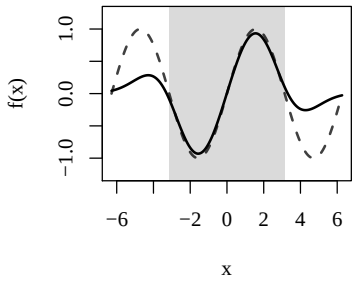


Figure 6.7: Support vector regression model fit of the sine function. The same graphical conventions as in Figure 6.4 have been used.

[18] Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2014). *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.6-4, http://CRAN.R-project.org/package=e1071 (retrieved 20.02.2015)

6.2.6 *Kriging*

Kriging is a spatial interpolation method originally developed in the field of geostatistics that was quickly adopted as a de-facto standard for meta- or surrogate modeling of computer experiments. In this application, a accurate but computationally demanding simulation model is approximated by a faster surrogate model to enable optimization in feasible time frames.

Kriging is a form of Gaussian process regression and as such, a Kriging Model (KM) can be used to predict means and variances, i.e., predictions are accompanied by error bars. For statistically sound predictions, the process generating the data should be deterministic. In other words, uncertainty should be epistemic, i.e., stem from incomplete data. See Sacks et al. [1989] for details and algorithms.[19]

Kriging is highly effective in generating accurate interpolation models for medium-dimensional data. As mentioned above, KM predictions include error bars, making the method robust against deployment errors and enabling optimization methods that balance exploration and exploitation in a near-optimal manner. Compared to the other regression techniques discussed above, fitting a KM is computationally expensive, as the requirement runtime scales as the cube of the number of data points, limiting the model to medium sized data sets, which also limits the maximum dimensionality of the data set.

Figure 6.8 shows a Kriging model fit of the sine function. The interpolative performance is excellent, with near zero prediction error. While model performance in the extrapolative region is bad, the fact that the model operates outside its trained region would be readily apparent by monitoring the associated error bars (not shown). All Kriging models used in this work were fitted via the R package `DiceKriging`.[20] See R's online documentation on the function `km` for details.

[19] Sacks, J., Welch, W., Mitchell, T., and Wynn, H. (1989). Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423

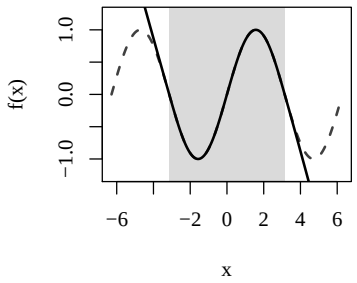


Figure 6.8: Kriging model fit of the sine function. The same graphical conventions as in Figure 6.4 have been used.

[20] Roustant, O., Ginsbourger, D., and Deville, Y. (2012). DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software*, 51(1):1–55. R package version 1.5.3, http://www.jstatsoft.org/v51/i01/ (retrieved 20.02.2015)

6.3 *Meta Models for Cyclone Dust Separators* (AppDust)

Cyclone separators are devices that utilize centrifugal forces of a vortex to separate particles from a gas or liquid stream, e.g., dust from flue gas. Their mode of operation is simple. A stream of particle-laden gas or liquid is channeled through an tangential

inlet pipe into a hollow cylinder, inducing a vortex where larger particles are pushed to the outer walls. Downwards, the cylinder tapers off to a conic part, where particles are siphoned. Clean gas or liquid leaves the cyclone through an immersion or outlet tube inserted into the cylinders lid. Figure 6.10 on page 154 shows a schematic. An example of a real-world cyclone dust separator is shown in Figure 6.9.

Cyclone separators have a broad spectrum of applications in multiple industries, ranging from power plants to vacuum cleaners. They can be build in multiple forms and sizes, ranging from centimeters to tens of meters in height. While other methods, such as mechanical or electrostatic filters, offer higher collection efficiencies, cyclone separators are energy-efficient, nearly maintenance-free and cost effective in both construction and operation. In contrast to many other filtering methods, they can be easily applied in high-pressure and high-temperature environments. In practice, different filtering methods are often applied in series, while cyclone separators offer an attractive solution for the first stage filtering device. Due to their principle of operation, cyclone separators have to be customized to a particular separation task in geometry and materials to provide satisfactory performance.

As in the previous case study, the RGP model induction process described in Section 3.2 provides the structural framework for the following subsections. The idea, data, and support for this case study is courtesy of Steinmüller Engineering GmbH, an engineering company specialized in power plant and environmental technology.[21]

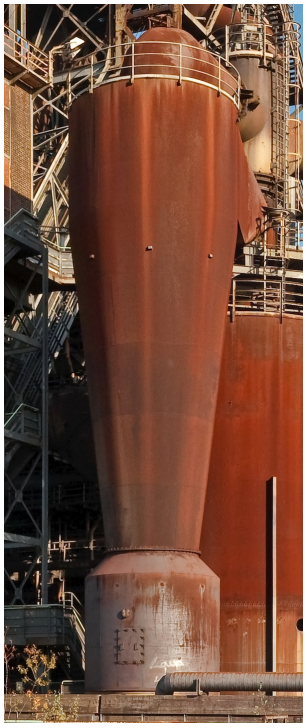


Figure 6.9: Large cyclone dust separator installed at blast furnace number five of the former Thyssen steelworks in Duisburg, Germany. Furnace five was build in 1973 and decommissioned after only 12 years of operation. Detail of an image retrieved from the Wikimedia Commons Media Archive at http://de.wikipedia.org/wiki/Datei:Hochofen_5.jpg.

[21] Data and R source code for this case study is available at https://rsymbolic.org/projects/ofdiss/repository/revisions/master/show/experiments/appDust.

6.3.1 *Modeling Project Setup*

Goal of this case study the development of surrogate models for geometry optimization of cyclone dust separators. As mentioned, the geometry parameters of a cyclone have to be customized to a particular separation task to provide satisfactory performance. Traditionally, this customization is performed on the basis of analytical models that have been specifically developed to predict key performance characteristics of cyclones. These characteristics contain the collection efficiency, which measures the share of particles removed from the input gas or liquid stream as a function of particle size, and the pressure drop, which measures the pressure loss caused by the cyclone. See Zaefferer et al. [2014] for a more detailed introduction to this topic.[22] While these simulation models are relatively simple, fast to execute on modern computers, and have been validated experimentally within the constraints of traditional geometries and operating conditions, there is a rising demand for simulation models that offer higher flexibility and accuracy. This demand is fueled by regulatory pressure for more environmentally friendly technology, which includes better cyclones. Today, multiphase computational fluid dynamics (CFD) methods are able to

[22] Zaefferer, M., Breiderhoff, B., Naujoks, B., Friese, M., Stork, J., Fischbach, A., Flasch, O., and Bartz-Beielstein, T. (2014). Tuning multi-objective optimization algorithms for cyclone dust separators. In *Proceedings of the 2014 conference on Genetic and evolutionary computation (GECCO 2014)*, New York. ACM Press

predict the collection efficiency and pressure drop of a cyclone with high accuracy for a wide range of operating conditions. The latter point is important, because modern cyclone applications require pressures and temperatures in regions where experimental data is scarce and the fidelity of traditional analytical models is questionable.

Cyclone models based on multiphase CFD models are computationally expensive, with compute time budget requirements of hours to days for a single cyclone configuration. CFD responds well to parallelization, allowing cyclone models to be simulated on compute clusters. The possible speedup is limited though. At the time of writing, these simulations still require dozens of minutes of compute time on a cluster of about 100 CPU cores.

With these compute time budget requirements, direct optimization based on CFD models would be infeasible. This is a fairly common problem, as many optimization tasks depend on computationally expensive computer experiments, e.g., CFD, finite element method (FEM) simulations, or Monte-Carlo experiments. The state-of-the art solution is to fit a regression model on a limited set of simulation results that are generated based on design of experiments methods. This model, known in this context as a *surrogate model* or *meta model*, is then used for optimization.

6.3.2 *Regression Problem Definition*

The key performance indicators for cyclones are collection efficiency (CE) and pressure drop (or pressure loss, PL). There are other possible criteria for optimization, such as material costs, manufacturing costs, or installation space requirements. For reasons of simplicity, these are omitted in this case study.

The AppDust problem can now be formulated as two independent regression tasks, given in R formula notation as

$$\begin{aligned} \text{CE} &\sim D_a + H + D_t + H_t + H_e + B_e + \text{PS} \\ \text{PL} &\sim D_a + H + D_t + H_t + H_e + B_e. \end{aligned}$$

Note that the particle size (PS) parameter is only relevant for the CE meta-model. Long names, domains, default values, and units for each of the parameters are given in Table 6.6. Only geometry parameters are targeted for optimization, therefore all process parameters except PS are held fixed at their default values. Figure 6.10 illustrates the cyclone geometry parameters. Parameter bounds and default values are taken from Zaefferer et al. [2014].

6.3.3 *Data Acquisition*

For fitting CE (PL) meta-models, a Latin Hypercube Design with 7×1024 (6×1024) points has been used. This comparatively large design for this 7 (6) dimensional problem was necessary to study the quality of the meta-model fits in detail. In an industrial setting, a much smaller design, probably with hundreds to about one

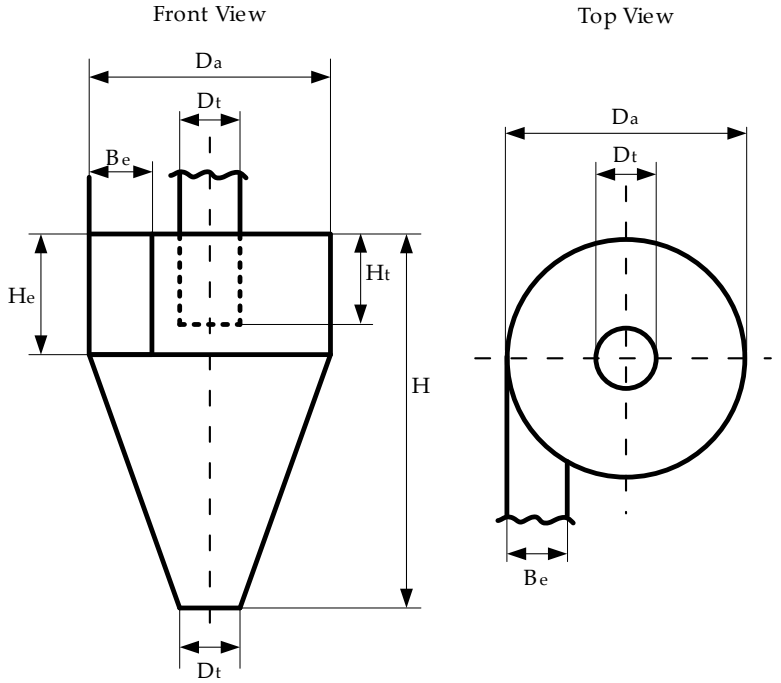


Figure 6.10: Cyclone geometry parameters subject to meta-modeling for later optimization. This figure has been adapted from Zaefferer et al. [2014].

Name	*Symbol*	*Domain*	*Default*	*Unit*
Geometry Parameters				
Cyclone diameter	D_a	$[1134, 1386]$	1260	mm
Cyclone height	H	$[2250, 2750]$	2500	mm
Outlet pipe diameter	D_t	$[378, 462]$	420	mm
Outlet pipe immersion	H_t	$[576, 704]$	640	mm
Inlet height	H_e	$[540, 660]$	600	mm
Inlet width	B_e	$[180, 220]$	200	mm
Process Parameters				
Viscosity	μ	-	$18.5 \cdot 10^{-6}$	$\mathrm{Pa} \cdot \mathrm{s}$
Flow rate	V_p	-	5000	m^3/h
Gas density	ρ_f	-	1.204	kg/m^3
Particle size	PS	$[0, 35]$	—	μm
Particle density	ρ_p	-	2000	kg/m^3
Particle concentration	c_e	-	50	g/m^3
Dependent Variables				
Pressure drop	PL	-	-	Pa
Collection efficiency	CE	-	-	%

Table 6.6: Independent and dependent variables for AppDust. Only geometry parameters are subject to meta-modeling in this case study. Other parameters are therefore held fixed at their default values. An earlier version of this table appeared in Zaefferer et al. [2014].

thousand points, would have been used used to keep the compute budget requirements of simulations within reasonable limits. The upper and lower parameter bounds are given in the domain column of Table 6.6.

As accurate CFD models for CE and PL were still in development at the time of this writing, the analytical models of Barth [1956] for CE and PL have been used as a provisional replacement.[23] Zaefferer et al. [2014] also contains a modern take on the details of these models.

[23] Barth, W. (1956). Berechnung und Auslegung von Zyklonabscheidern auf Grund neuerer Untersuchungen. *Brennstoff-Wärme-Kraft*, 8(1):1–9

Calculating the analytical models for CE and PL on their respective Latin Hypercube Designs yields the two data sets used in this case study. Each of this data sets is then divided equally into training and validation sets by random sampling without replacement. As the validation set uses the same parameter bounds as the training set, model extrapolation quality is not measured in this case study.

6.3.4 *Data Preprocessing*

All meta-models examined in this case study were fitted to training data and evaluated on validation data without special preprocessing. As the data has been artificially generated by means of analytical models applied within sensible input bounds, no numerical problems or missing values had to be handled. Fractional efficiency lattice plots were generated to ensure that the response of the analytical models is within expectations during the EDA phase. A selection of these plots will be shown in Section 6.3.7.

6.3.5 *Linear Modeling*

In addition to LM, Kriging, MARS models, PCR models, RF models, and SVM models were additionally applied to the AppDust data. In a typical industrial setting, only a subset of these experiments would be feasible. Also in contrast to real-world practice, for each meta-modeling technique that involves randomized algorithms, ten independent fits were generated and tested.

6.3.6 *Symbolic Regression*

Symbolic regression was performed using RGP with the GMOGP-FCA search heuristic using default parameter settings from Table 2.4. See Section 2.9.2 for details. The default function set $\{+, -, *, /, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$ has been used. The compute time budget for evolving each of the meta-models has been set to 50 million candidate solution evaluations. Ten repeated runs were performed to produce statistically reliable results.

6.3.7 *Model Validation*

To access the accuracy of the meta-models generated, MAE values on validation data have been calculated. Additionally, CE meta-models can be inspected visually by means of fractional efficiency lattice plots to examine meta-model response in detail.

Collection Efficiency A statistical summary of the validation MAE values of all meta-models evaluated in this study are shown in Table 6.7 and visualized as Box plots in Figure 6.11. For RGP, the solution with best training MAE has been selected from the Pareto front of solutions in each of the ten runs. Meta-models are also ranked by validation MAE, where lower MAE values are better. The KM meta-model achieves near perfect results with validation MAE values close to zero and very low, but non-zero variance between the ten independent fits. This result asserts the role of Kriging as the de-facto standard for simulation meta-modeling.

Regarding the remaining regression techniques, the RGP implementation of the GMOGP-FCA and RF models yield the best results. RGP achieves a slightly better median with higher variance than RF, while the RF results exhibit much lower variance. MARS, ranked fourth based on validation MAE median, offer an interesting option for their comparatively low compute time requirements and deterministic behavior, i.e., zero variance between the ten independent fits. SVM model fits are also still acceptable, but could probably be improved by parameter tuning. The linear meta-modeling techniques LM and PCR fail on this task because of the inherent non-linearity of the AppDust problem.

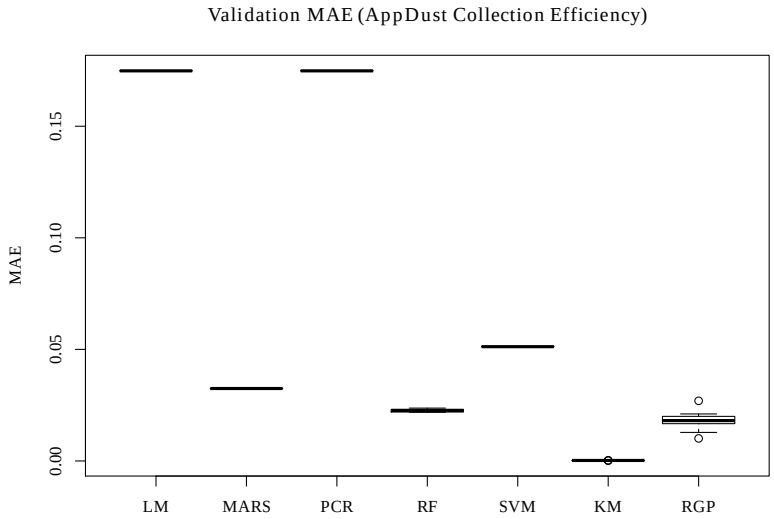


Figure 6.11: Validation MAE for AppDust collection efficiency of several regression models, compared to validation MAE of the best model found by RGP.

Figures A.13 and A.14 in Appendix A show lattice plots of the fractional efficiency for cyclones of default geometry but varying in-

		LM	*MARS*	*PCR*	*RF*	*SVM*	*KM*	*RGP*
MAE	*Median*	0.1748	0.0325	0.1748	0.0227	0.0512	**0.0002**	0.0181
	SD	**0.0000**	**0.0000**	**0.0000**	0.0006	**0.0000**	**0.0000**	0.0045
Rank		6	4	6	3	5	**1**	2

Table 6.7: Validation MAE for AppDust collection efficiency of several regression models. Best values are marked with bold font. Results for randomized algorithms are based on ten samples. Values are rounded to four decimal places.

let height (H_e), as well as meta-model predictions based on the RGP and SVM, as well as on the RGP and RF meta-models, respectively. Each of the nine subplots shows the fraction of particles collected for particle diameters between 0 and 35 μm as predicted by the analytical model (solid gray line), the SVM or RF model (dotted black line), and the RGP model (dashed black line). The subplots differ in (H_e), which is varied from 540 to 660 mm (bottom left to top right subplot). All plots are based on validation data. Based on visual inspection, the RGP meta-model provides a very good fit of the validation data. The RF meta-model fit worsens at the upper and lower bounds of the H_e parameter, while the SVM meta-model fails to capture the collection efficiency roll-of for small particle sizes and erroneously predicts negative collection efficiency values in this region.

Pressure Drop As with the CE meta-models, a statistical summary of the validation MAE is compiled to assess the relative accuracy of all meta-models evaluated in this study. This summary is shown in Table 6.8 and visualized as Box plots in Figure 6.12. As in the previous result analysis, the RGP solution with best training MAE has been selected from the Pareto front of solutions in each of the ten runs. Again, the KM meta-model achieves validation MAE values close to zero with very low variance.

The RGP implementation of the GMOGP-FCA achieves the second best median validation MAE of 5.0890, although with relatively high variance. From here on, the result ranks begin to differ from the previous ranking. The SVM meta-models, ranked third, exhibit a significantly worse median validation MAE of 17.6757. MARS fares even worse, followed by LM and PCR that share the fifth rank.[24] The RF meta-model ranks last with an even worse median validation MAE then that of the linear models. This result is surprising, as the same meta-modeling technique ranked fourth based on median training MAE.

[24] During PCR model fitting, all principle components were selected, rendering the PCR meta-model equivalent to the LM meta-model.

		LM	*MARS*	*PCR*	*RF*	*SVM*	*KM*	*RGP*
MAE	*Median*	28.8141	22.3014	28.8141	40.5395	17.6757	**0.0108**	5.0890
	SD	0.0000	**0.0000**	**0.0000**	0.3166	**0.0000**	**0.0000**	10.5661
Rank		5	4	5	6	3	**1**	2

Table 6.8: Validation MAE for AppDust pressure drop of several regression models. The remarks of Table 6.7 also apply here.

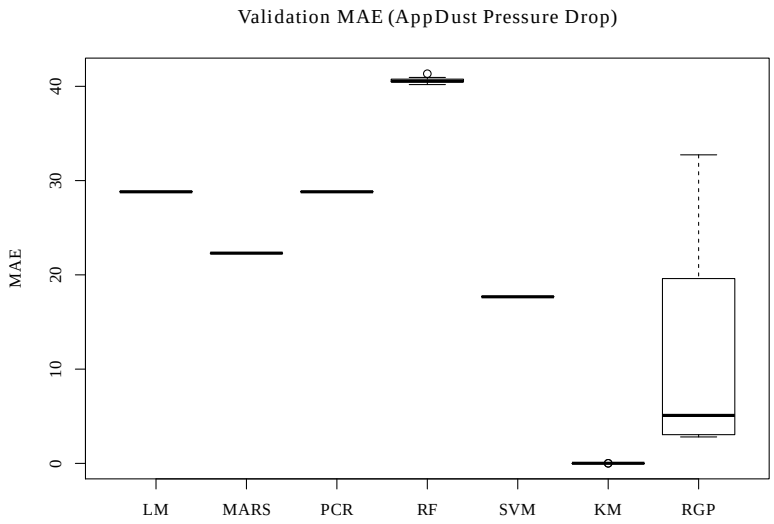


Figure 6.12: Validation MAE for AppDust pressure drop of several regression models, compared to validation MAE of the best model found by RGP.

6.3.8 *Model Deployment*

Based on the model validation results, either the KM or RGP meta-models would probably be deployed in practice, depending on whether accuracy or understandability are considered more important. As all meta-models of this case study are based on synthetic data, real-world model deployment was postponed until meta-models based on CFD simulation results become available.

6.4 *Roll Train Control Models* (AppSteel)

Rolling is a metal forming process for thickness or width reduction. Metal stock is passed through one or more pairs of rolls. More formally, according to Wagoner and Chenot [2005],

> the rolling process can be defined as a continuous process of plastic deformation for long parts of constant cross section, in which a reduction of the cross sectional area is achieved by compression between two rotating rolls (or more).[25]

[25] Wagoner, R. and Chenot, J. (2005). *Metal Forming Analysis*. Cambridge University Press, New York

The assembly of rolls and supporting infrastructure is known as roll train. Rolling at temperatures below recrystallization of the processed metal is known as cold rolling, while rolling above recrystallization temperatures is known as hot rolling. At the time of this writing, hot rolling is the most important metal forming process in terms of tons of material processed. Similarly, cold rolling is the most important cold metal forming process in terms of material throughput.

Various geometric shapes can be produced by rolling. In flat rolling, metal slabs or sheets of thicknesses between several centimeters down to tenths of millimeters are produced by rolling with flat cylinders. Shape rolling allows the production of more complex

geometry, such as beams or rods. As the required transformations in geometry are often too large for a single pass, most rolling processes consist of multiple passes that are either applied successively with the same rolling station or continuously with multiple rolling stations.

In this case study, the focus is on a flat hot rolling process used in steel mills. This section is organized much like the last case study on cyclone geometry optimization and follows the framework introduced in Section 3.2. The idea and supporting data for this case study was kindly provided by a large manufacturer of roll trains and metalwork technology.[26]

[26] Data and R source code for this case study is available at `https://rsymbolic.org/projects/ofdiss/repository/revisions/master/show/experiments/appSteel`.

6.4.1 *Modeling Project Setup*

Goal of this case study is the development of models for predicting steel slab width reduction caused by a two pass flat rolling process based on measurement data. Figure 6.13 shows a simplified drawing of the process under study. The rolling station shown consists of an edger, meaning a pair of vertical rolls for width reduction, followed by rougher, meaning a pair of horizontal rolls for thickness reduction, followed by a width gauge. The metal product, composed of a specific steel alloy, is threaded through the station in multiple passes, back and forth, until the desired thickness and width reduction is attained. Width can only be measured accurately after odd passes, where the product moves from left to right, while the process can only finish after an even amount of passes. The edger can only be used in odd passes.

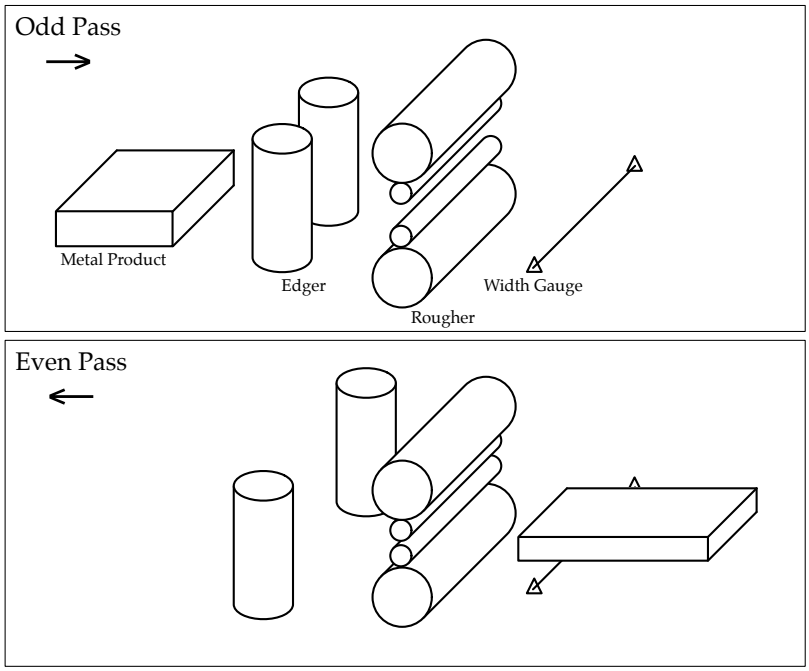


Figure 6.13: Schematic of the steel rolling process studied in the AppSteel case study. The rolling station consists of a pair of vertical rolls for width reduction (edger), followed by a pair of horizontal rolls for thickness reduction (rougher). The metal product is processed in multiple back and forth passes. Width can only measured after an odd pass, while the process can only finish after an even pass. The edger can only be used in odd passes. The upper half of the drawing shows an odd (left to right) pass, the lower half shows an even (right to left) pass of the process.

Accurate models of the width reduction obtained after a pair of

passes are of high interest, because metal products are produced to exact specifications of minimum width and thickness. Batches that fall short of minimum width or thickness have to be discarded as scrap. On the other hand, excess width or thickness is not paid for, causing unnecessary costs. Steel mill operators, operating under high economic pressure, therefore have a high interest in meeting product specifications as exactly as possible, without undershooting.

The models are to be deployed as predictive control models, meaning that a model is part of an optimization loop, where the model or process parameter input space is continuously searched for settings that robustly achieve the desired width reduction, which can change throughout a production batch. As the time required for two rolling passes is measured in seconds, model evaluation must be fast.

The physics of hot metal forming is complex and contains highly non-linear effects. The width reduction of a steel slab caused by the two pass process described here is expected to depend on a large number of parameters, including width and thickness set points, as well as steel alloy composition. Recently, non-linear FEM based models have been developed for simulating the rolling process with high accuracy. These models are computationally expensive and are complex to setup and parameterize, though. At the time of writing, this disqualifies these models for most uses in predictive control for steel rolling.

Custom analytical models devised by domain experts, as well as data-driven models, offer two alternatives that are used in practice. In this case study, multiple data-driven modeling approaches are evaluated.

6.4.2 *Regression Problem Definition*

Creating data-driven models for width reduction in steel rolling can be formulated as a regression problem. Using R formula notation, the AppSteel problem is given as

$$\begin{aligned} Y \sim\ & \mathrm{PMEAS} + \mathrm{AWC} + \mathrm{THICK} + \mathrm{THICKRED} + \mathrm{PTHICKRED} \\ & + \mathrm{ZR} + \mathrm{W} + \mathrm{C} + \mathrm{SI} + \mathrm{MN} + \mathrm{P} + \mathrm{S} + \mathrm{CR} + \mathrm{MO} + \mathrm{NI} \\ & + \mathrm{V} + \mathrm{AL} + \mathrm{CU} + \mathrm{CO} + \mathrm{TI} + \mathrm{NB} + \mathrm{N} + \mathrm{SN} + \mathrm{AS} + \mathrm{B}. \end{aligned}$$

The parameter symbols used in this formula are explained in Table 6.9. The steel alloy composition is given in the form of 20 parameters named after their respective chemical element symbol. Other independent variables to be used as model inputs include the measured width at the previous odd pass, the width set at the edger, the thickness of the material before rolling the current pass (input thickness), the thickness reduction set at the rougher, and the measured thickness reduction caused by the previous even pass. In total, the AppSteel regression problem has 25 independent variables. The only dependent variable, width reduction after the next even

pass, can attain negative values. This happens because thickness reduction via flat rolling naturally entails lateral expansion of the material.

Name	*Symbol*	*Domain*	*Unit*
Independent Variables			
Width at previous odd pass	PMEAS	$[750, 2000]$	mm
Width set at edger	AWC	$[750, 2000]$	mm
Input thickness	THICK	$[40, 300]$	mm
Alloy composition	(20 Chemical symbols)	$[0.0, 1.0]$	%
Thickness reduction set at rougher	THICKRED	$[0, 50]$	mm
Thickness reduction of previous even pass	PTHICKRED	$[0, 50]$	mm
Dependent Variable			
Width reduction after next even pass	Y	$[-50, 25]$	mm

Table 6.9: Independent and dependent variables for AppSteel. The alloy composition is represented as 20 parameters named after their respective chemical symbols and condensed into one line in this table. AppSteel is a regression problem with 25 dimensions.

6.4.3 *Data Acquisition*

All models evaluated in this case study have been fitted on data provided externally. The dataset is based on a Latin Hypercube Design with 6190 points. Width reduction values for each of these points were generated by a proprietary simulator not available to the author. The data was then divided into training and validation data sets by random sampling without replacement. One tenth of the data has been used for training (619 points), nine tenths (5571 points) has been used for validation, as proposed by the industry partner. Training and validation data use the same parameter bounds, therefore model extrapolation quality is not evaluated.

The use of simulated data for model evaluation is common practice at the industry partner, as controlled experiments with steel roll trains are too costly and time consuming to perform in the initial stage of comparing multiple modeling techniques.

6.4.4 *Data Preprocessing*

As in the previous case study, all models under evaluation were fitted to training data and evaluated on validation data without any data preprocessing. The data has been generated by a simulation model applied within specified parameter bounds, and no numerical problems or missing values had to be accounted for. Before modeling, the data has been visually inspected through scatter plots and correlation diagrams.

6.4.5 *Linear Modeling*

In addition to LM, the same set of regression modeling techniques as in the previous case study described in Section 6.3 was evaluated. For each meta-modeling technique that uses randomized algorithms, ten independent fits were generated and evaluated.

6.4.6 *Symbolic Regression*

As in the previous case study, symbolic regression was performed using RGP configured with the GMOGP-FCA search heuristic using default parameter settings from Table 2.4. See Section 2.9.2 for details on the parameterization. Also, the default function set $\{+, -, *, /, \sin, \cos, \tan, \exp, \log, \sqrt{x}\}$ has been used. Each RGP run had a compute time budget of 50 million candidate solution evaluations. Also as in the previous case study, ten repeated runs were performed to produce statistically reliable results.

6.4.7 *Model Validation*

MAE values on validation data have been calculated to access the accuracy of the different modeling techniques evaluated in this case study. Table 6.10 shows a statistical summary of the validation MAE values of all models evaluated. These values are visualized in the form of Box plots in Figure 6.14. The solution with best training MAE has been selected from the Pareto front of solutions in each of the ten RGP runs. The resulting models are ranked by median validation MAE, where lower MAE values are better.

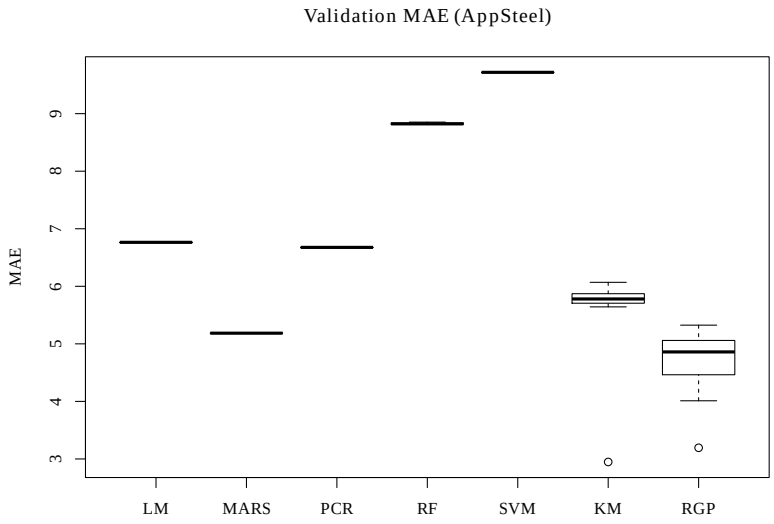


Figure 6.14: Validation MAE for AppSteel of several regression models, compared to validation MAE of the best fitting models found by each of ten independent runs.

		LM	*MARS*	*PCR*	*RF*	*SVM*	*KM*	*RGP*
MAE	*Median*	6.7639	5.1853	6.6760	8.8225	9.7177	5.7800	**4.8597**
	SD	**0.0000**	**0.0000**	**0.0000**	0.0134	**0.0000**	0.9136	0.6404
Rank		5	2	4	6	7	3	**1**

Table 6.10: Validation MAE for AppSteel of several regression models. The remarks of Table 6.7 also apply here.

According to this ranking, the models created by RGP perform best with a median validation MAE of 4.8597. MARS models are slightly worse, with a median validation MAE of 5.1853, but

nonetheless offer an attractive alternative here because of their deterministic nature (zero variance in the results achieved). KM models rank third, with an median validation MAE of 5.7800. Both the RGP and KM run results contain a single outlier each with very low validation MAE values of 2.9480 (KM) and 3.1943 (RGP). The box plots reveal that, with these outliers removed, KM validation MAE result variance is lower than RGP validation MAE result variance, consistent with the results of the previous case study.

PCR models, ranked fourth with a median validation MAE of 6.6760, achieve slightly better results than LM models, that reach a median validation MAE of 6.7639. Inspection of the PCR models reveals that only five out of a maximum of 25 possible principal components were selected to be included in the model. RF and SVM fail to produce good results for this particular application with median validation MAE values of 8.8225 (RF) and 9.7177 (SVM), ranking sixth and seventh. These results cannot be explained by these models failing to achieve good fits on the training data using their respective parameterizations. Ranked by median training MAE, RF models take the second place, with only KM reaching better median training MAE values. SVM models take the fifth place, with better median training MAE values than both PCR and LM.

RGP Model Analysis The following paragraphs focus on tools and techniques for analyzing GP models available in RGP. As ten repeated runs have been conducted in this case study, the first run, based on random number generator seed 1, has been chosen arbitrarily for deeper analysis. In practice, probably only a single RGP run would have been conducted due to compute time budget constraints.

As GMOGP-FCA is a multi-objective GP search heuristic, a typical starting point for analyzing the set of generated solutions is a Pareto front plot. Figure 6.15 shows this plot. The x-axis indicates training fitness, measured as SMSE, the y-axis indicates solution complexity, measured as visitation length. Individual solutions are shown as black points and numbered by rank based on training fitness. The Pareto front approximation represented by this set of solutions is shown as a thick gray line.

Table 6.11 presents the same data in a textual format. There are some interesting observations to be made that, based on experience, apply to most result Pareto fronts generated by GMOGP-FCA. First, solution complexity and solution accuracy are to be conflicting goals. Second, there is an exponential relationship between solution complexity and solution accuracy. A linear model based on the R model formula $\log(\text{Training Fitness}) \sim \log(\text{Complexity})$ achieves an adjusted R^2 of 0.9641, i.e., a very good fit, when fitted on the data reproduced in Table 6.11.

This clearly indicates a point of diminishing returns regarding the relationship between model complexity and model accuracy, suggesting that the best trade-off between these conflicting ob-

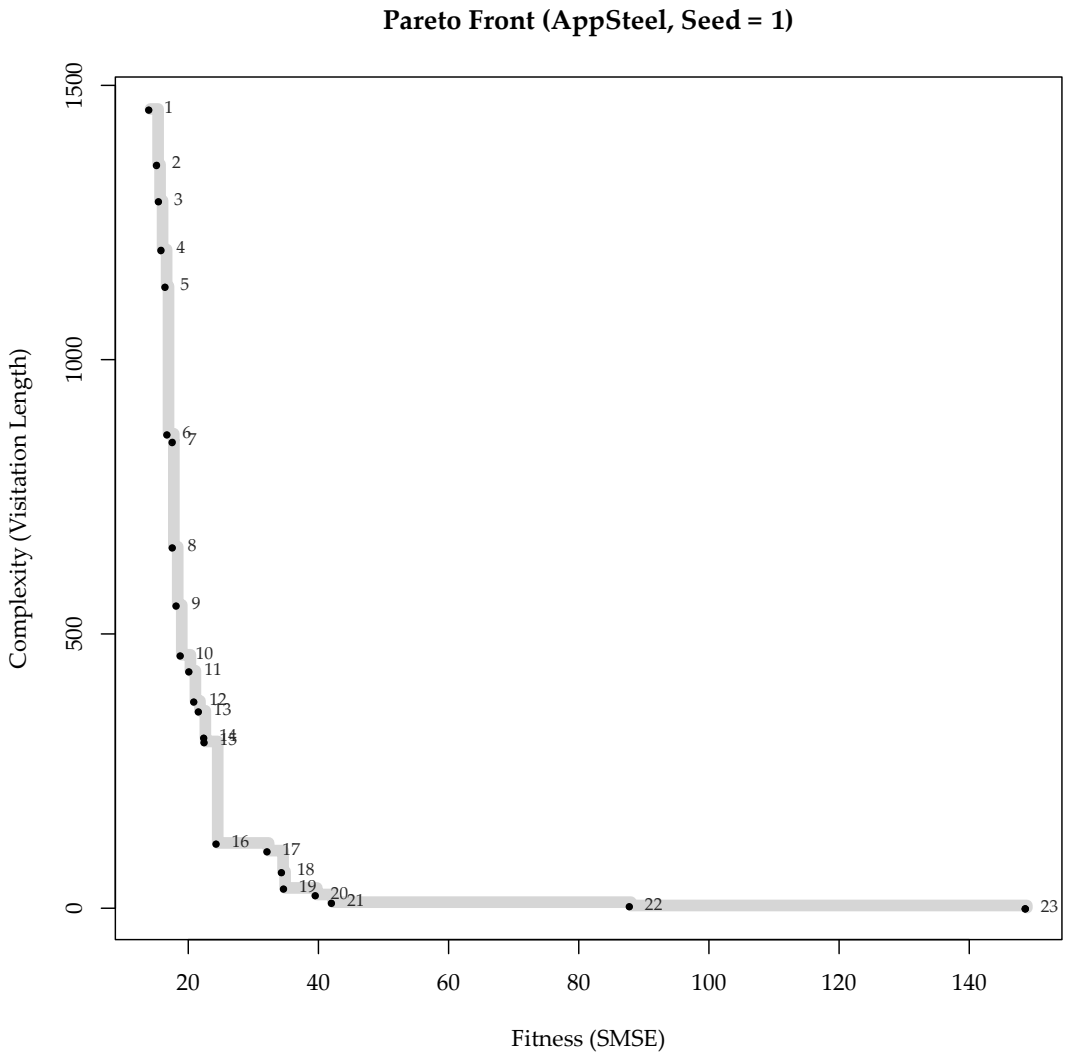


Figure 6.15: Pareto front plot of the AppSteel RGP run with random seed 1. Solution training fitness is shown on the x-axis, while solution complexity is shown on the y-axis. Individual solutions are shown as black points and numbered by rank based on training fitness (SMSE). Training fitness is used instead of validation MAE, because this plot has been generated by RGP during a GP run. The Pareto front approximation is shown as a thick gray line. See Table 6.11 for a tabular representation and additional details.

jectives should be found near the knee of the Pareto front. See Section 5.5.4 for a formal definition of this concept. In the case discussed here, the knee of the Pareto front lies at solution 16, which attains a validation MAE of 4.18 with a modest visitation length (complexity) of 119. This model, given as

$$Y = (\text{PMEAS} - \text{AWC}) \cdot \cos(\sqrt[4]{\text{THICK}}) \cdot \frac{\text{AWC} \cdot \sqrt{\text{PMEAS}}}{\text{THICK}}$$

is simple enough for human analysis. Note that, due to use of SMSE fitness, the scale of the output may be off by a constant factor, which can be easily corrected. Solution 21 provides an even simpler model, that attains a validation MAE of 5.2 with a low visitation length of 11:

$$Y = \frac{\text{PMEAS} - \text{AWC}}{\text{THICK}}$$

Also note that solutions 16 and 21 shown in the previous paragraph, which are near the Pareto front knee, only contain the input

	Training Fitness (SMSE)	*Complexity (Visitation Length)*	*Training* MAE	*Validation* MAE
1	14.18	1457	2.98	3.19
2	15.37	1356	3.06	3.29
3	15.67	1290	3.08	3.30
4	16.05	1201	3.12	3.33
5	16.67	1134	3.17	3.37
6	16.96	865	3.24	3.40
7	17.77	851	3.32	3.48
8	17.79	659	3.33	3.52
9	18.38	553	3.40	3.59
10	19.00	462	3.42	3.60
11	20.33	433	3.48	3.66
12	21.10	378	3.67	3.83
13	21.80	360	3.64	3.78
14	22.62	312	3.64	3.85
15	22.68	304	3.64	3.86
16	**24.52**	**119**	**4.02**	**4.18**
17	32.35	105	4.43	4.66
18	34.57	67	4.50	4.67
19	34.88	37	4.52	4.68
20	39.76	25	4.65	4.94
21	42.25	11	4.93	5.20
22	88.02	5	7.24	7.38
23	148.85	1	9.95	9.97

Table 6.11: Pareto front after 50 million fitness evaluations on AppSteel, RGP with GMOGP-FCA search heuristic, random seed 1. Table rows are sorted by training fitness (SMSE). The Pareto front knee is marked with bold font. See Figure 6.15 for a graphical representation of this data.

variables PMAES, AWC, and THICK, i.e., only three out of 25 possible independent variables. This observation hints at the possibility of spurious independent variables. To investigate, the occurrences of each input variable in each model of the result population is counted. Variable occurrences in Pareto front models are counted separately. The result of this census is visualized in Figure 6.16 in form of a bar plot. In this plot, input variable occurrences in models that are members of the Pareto front are plotted as black bars, all other occurrences are plotted as gray bars.

Plots similar to Figure 6.16 where introduced by Kotanchek et al. [2010].[27] They also illustrate the potential of multi-objective GP as a feature selection method.

[27] Kotanchek, M. E., Vladislavleva, E., and Smits, G. F. (2010). Symbolic regression via genetic programming as a discovery engine: Insights on outliers and prototypes. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation Series, pages 55–72. Springer, New York

Only four independent variables are prevalent in the models of the result population, that is, in order of decreasing frequency, PMAES, AWC, THICKRED, and THICK. Input variables pertaining to the material properties are nearly completely absent, together with the input variable PTHICKRED that describes the thickness reduction caused by the previous even pass. This data supports the hypothesis of spurious variables in the model training data sets and also indicates what these variables are. The observation that only five principal components were selected during PCR model generation further supports this hypothesis.

As the training and validation data sets were generated through a proprietary simulator, it seems plausible that the simulation either omits the influence of material properties or that material prop-

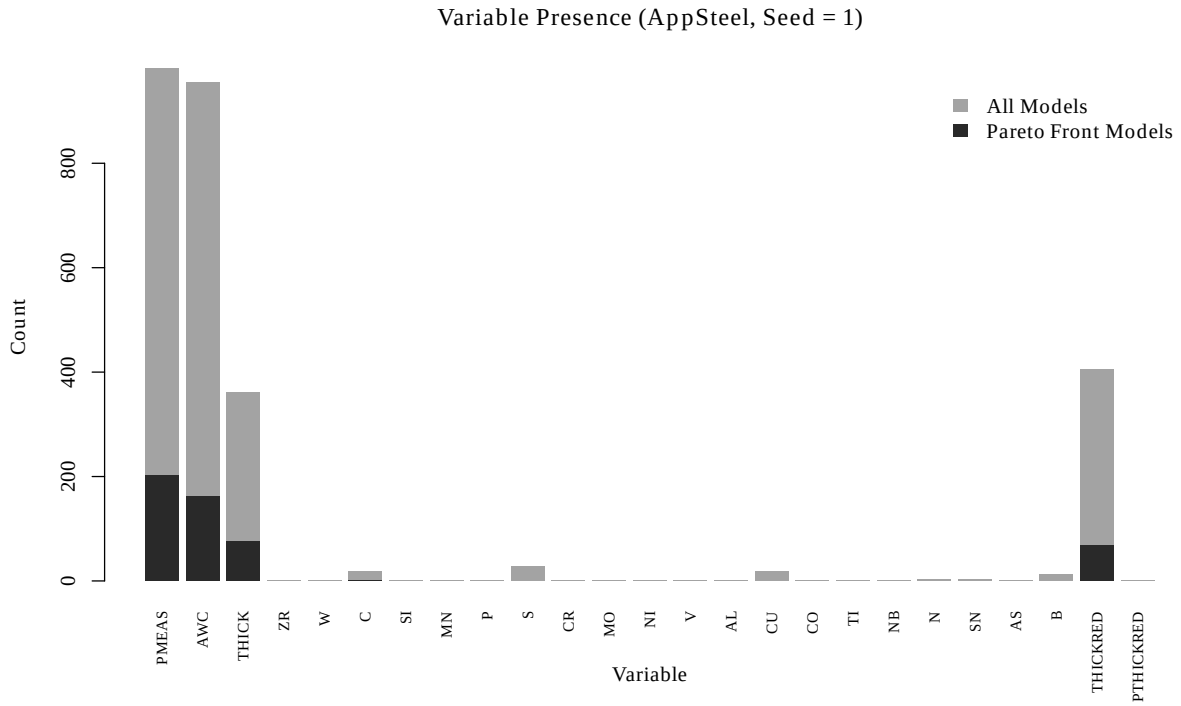


Figure 6.16: Number of input variable occurrences among the models in the result population of the AppSteel RGP run with random seed 1. Variable occurrences in models that are members of the Pareto front are plotted as black bars, all other occurrences are plotted as gray bars.

erties are not influential to the width reduction to be modeled, at least in the parameter bounds studied. The same applies to the independent variable PTHICKRED.

6.4.8 Model Deployment

On the basis of the results presented in the last subsection, either the models generated by RGP or MARS would be deployed. Both models can be evaluated efficiently enough for use in predictive control. As all models were fitted on simulated data only, a second evaluation based on real-world data generated by means of controlled experiments would be advisable to ensure correct model behavior in production usage.

6.5 Conclusions

This chapter put the performance of the GP algorithms implemented in RGP into context by comparisons with other state-of-the-art GP systems based on well-known artificial test problems. Furthermore, two extensive real-world applications were described in detail, in the form of case studies, demonstrating how RGP can be used effectively in practice. Comparisons with alternatives to GP, including methods from machine learning and classical statistics, were also part of these case studies.

Judged by median validation MAE, RGP was able to consistently generate good models for both artificial and real-world tasks. Over-

all, no other general regression technique provided consistently better results. Analysis of the real-world case studies establishes Kriging (KM) as a viable alternative to RGP that offers a different set of trade-offs. KM offer the important benefit of providing error estimates together with predictions, while RGP models are white-box models open to human interpretation, at least to a certain degree of model complexity. Depending on the dimensionality of the input data, KM are about equally computationally expensive to fit as RGP models. KM have an advantage at lower dimensions, while RGP models tend to perform better at higher dimensions, due to their inherent variable selection capability. As a downside, the variation in accuracy of models generated in independent RGP runs was distinctly larger than with other regression techniques incorporating randomized algorithms. There are at least two approaches to alleviate this problem, none of which implemented in RGP at the time of writing. First, ensemble techniques such as bootstrap aggregating can be applied, with the cost of loosing the simplicity and understandability of a single solution formula associated with GP. Second, the results of multiple parallel runs can be combined, lowering result variance at the cost of a higher required compute time budget.

Model accuracy of the other models evaluated during the real-world case studies was notably less consistent and much more dependent on the problem at hand. For example, RF models reached rank three based on median validation MAE when predicting AppDust cyclone collection efficiency, but only rank six when predicting cyclone pressure drop. SVM models showed similar problems. Naturally, both RF and SVM results could be improved by expert tuning, but this is not the point to be made in this study. In the hands of a user with detailed domain knowledge, some statistical background, but no specialized training in any of the regression techniques discussed, RGP seems to provide high accuracy models consistently. Cortez [2014] provides two additional examples for the successful application of RGP to regression and time series forecasting. He concludes his example on time series forecasting, which revolves around the accurate prediction of sunspot numbers, with the following remarks:

> Both ARIMA and genetic programming predictions are close to the true sunspot values. Overall, the genetic programming solution produces slightly better forecasts with an improvement of 0.7 when compared with the ARIMA method in terms of MAE measured over the out-of-samples. This is an interesting result, since ARIMA methodology was specifically designed for [time series forecasting] while genetic programming is a much more generic optimization method.[28]

[28] Cortez, P. (2014). *Modern optimization with R*. Use R! Series. Springer International Publishing, Cham, Switzerland

7
Summary and Outlook

The previous chapters described a formal framework for evolutionary algorithms in general and GP in particular, detailed the design and implementation of RGP, a modular GP system built on this framework, and showed how this framework is combined with the framework of SPO to research GP operators and search heuristics while producing reliable and reproducible results. Finally, artificial test problems and two case studies from different industries were employed to demonstrate how RGP can be used successfully in practice.

In this chapter, the results of this work are summarized and put into context (Section 7.1). The original contributions presented in Section 1.2 are revisited in light of these results (Section 7.2). Open questions are listed in Section 7.3 as opportunities for further research and possible extensions to RGP, designed to advance adoption of GP for practical applications, are presented in Section 7.4.

7.1 Summary

GP is the application of evolutionary computing to the problem of discovering symbolic expressions that are near-optimal according to one criterion, or in the case of multi-objective GP, near-optimal compromises according to multiple criteria. Chapter 2 started with a short and informal introduction to GP to build intuition by reflecting on the biological metaphors inspiring evolutionary computing and by giving application examples. The chapter proceeded with the introduction of a formal framework for describing evolutionary algorithms, particularly GP, in the abstract, without referring to concrete implementations. This approach has multiple benefits. It provides a framework for structuring modular GP implementations. It also provides the vocabulary to describe GP operators, e.g., for variation and selection, improving understanding and facilitating concise and correct implementation. Finally, linking theory and implementation by basing both on the same formal grounding, simplifies co-evolution of theory and practice, i.e., adapting an implementation to quickly exploit new theoretical knowledge. This last point will be illustrated in Section 7.4, where extensions to the RGP systems will be proposed based on current

research. Chapter 2 concluded with the introduction of GMOGP, a modern multi-objective GP search heuristic.

Building on the formal framework of Chapter 2, Chapter 3 introduced RGP, a modular GP system integrated in the R environment for statistical computing. This system also provided the basis for the experimental studies on GP system parameterization described in this work. The system implements classical untyped tree-based GP, as well as strongly typed GP, and Pareto GP, in a modular fashion. It allows direct customization and replacement of every algorithm component, facilitating rapid experimentation. RGP's feature set compares favorably to the feature sets of other GP systems, both commercial and open source.

Chapter 3 also provided a process for model induction with RGP that covers the important tasks of data import, preprocessing, result analysis, model validation, and model deployment, as a roadmap for practitioners. In addition to documenting individual features, RGP was introduced by means of tutorial examples. RGP's principal design goals of effectiveness, approachability, extensibility, modularity, and efficiency were defined and put into context of the system's implementation. Finally, the design rationale and implementation of a modern web-based user interface for symbolic regression with RGP has been explained in detail.

Chapter 4 marked the beginning of the empirical part of this thesis. After introducing the framework of SPO and its implementation SPOT, two approaches to scalable test functions were introduced. Spurious variable test functions can be scaled in difficulty by introducing additional uncorrelated independent variables to conceal the functional dependency between driving (independent) variables and the dependent variable. While conceptually simple, these test functions only provide a single variant per difficulty level, which can introduce overfitting problems into the SPO process.

Kriging-based scalable random test functions were then introduced to alleviate this problem. Here, a Kriging model is fitted on samples of a base test function, whose parameters are than perturbed to provide arbitrary many test function instances. While this approach can be combined with spurious variables, it also offers some control over the condition number of the generated test function instance, providing another means of scaling test function difficulty.

The main empirical work conducted in this thesis revolved around three scientific claims, that were broken down into testable statistical hypotheses and examined in Chapter 5. The first claim, stating that complex problems require complex algorithms, seems to be valid. GP system performance is reversely correlated to problem difficulty, and modern GP algorithms consistently outperform simpler GP algorithms.

The second claim, stating that parameter settings of modern GP systems can be effectively tuned, also holds. It was shown, for the GP system used in this work, that there exist parameters with significant effect on system performance and that there are significant differences in the relative effect of the parameters studied. Furthermore, it has been shown that GP system parameter tuning is effective in improving result quality.

The third and final claim, stating that tuned GP system parameter settings are robust within a problem class, appears to hold up to first empirical tests. The existence of problem classes where GP performance can be measurably improved by SPO has been shown. Pairs of problem classes with measurable differences in their SPO-tuned parameter settings were demonstrated. These results hint at the robustness of tuned parameters within a problem class, while additional experimentation with different problem classes would be required to secure this result.

Based on the parameter tuning process described in Chapter 5, best practices for GP system parameter tuning have been proposed. GP function set, error measure, and population size were identified as the parameters with highest expected effect on GP system performance.

Chapter 6 demonstrated the relative performance of RGP on well-known test problems by means of comparisons with other state-of-the-art GP systems. Additionally, two extensive real-world case studies were described in detail. These case studies also contained comparisons with alternatives to GP for regression, such as methods from machine learning and classical statistics. RGP was able to consistently achieve good results for both artificial and real-world tasks. Overall, no other general regression technique provided consistently better results. The variation in accuracy of models generated in independent RGP runs was distinctly larger than with other regression techniques that incorporate randomized algorithms. Approaches to alleviate this problem were proposed. In conclusion, RGP using the multi-objective GMOGP search heuristic consistently provided high accuracy models without requiring specialized expert knowledge for model setup. It is highly probable that these results could be improved even further by SPO.

7.2 *Original Contributions*

As mentioned in Section 1.2, this work makes three contributions to the state-of-the art in GP systems:

- *A Modular GP System* The design and implementation of RGP, a modular GP system based on the R environment for statistical computing has been presented in detail (cf. Chapter 3).
- *A Comprehensive Empirical Analysis of Modern GP Heuristics* Based on RGP and SPO, a framework for reproducible empirical re-

search in GP has been developed (cf. Chapter 5).

- *New Industrial Applications for GP* Finally, RGP configured with the multi-objective GMOGP search heuristic has been applied to two real-world industrial applications from mechanical and process engineering (cf. Chapter 6).

See Section 1.2 for additional details on each of these contributions. Methodological contributions of this work include the free availability of all source code and experiment data[1] to facilitate reproducible research, as postulated in the Science Code Manifesto[2], as well as the consequent use of, and integration with, free and open source software to make all algorithms and techniques developed available to a wide audience.

[1] see http://rsymbolic.org

[2] Barnes, N. et al. (2013). Science code manifesto. http://sciencecodemanifesto.org/ (retrieved 20.02.2015)

As already stated in the introduction, the primary motivation of this work was to provide a practical system applicable to real-world applications. RGP constitutes a successful first attempt at such a system. It proved to be already useful not only for academic research, but also for multiple practical applications.

7.3 Open Questions

While GP seems to gain relevance in academic and industry application, there are both perceived and real roadblocks preventing wider adoption. This section lists some of the most common objections to GP, suggests possible solutions already available today or directions for future research.

A typical objection against the practical use of GP is perceived inefficiency of evolutionary search in general. This objection is only valid when the general structure of the solution is known, which seems to be often the case, hence the success of (generalized) linear models. There might be a selection bias here, as stories of the unsuccessful application of regression methods often remain untold. If the solution structure is not known and only insufficient resources are available for manual analysis, GP is often able to produce good solutions that rival domain-specific methods and are open to human analysis and understanding. This assertion is supported by the results presented in Chapter 6.

A objection to using RGP in practice is a perceived lack of well-known techniques of securing solution quality. Standard techniques from machine learning, such as cross-validation, can be used to alleviate this problem. In the case of regression analysis, techniques developed for generalized linear models (GLM) are sometimes applicable to models discovered by evolutionary search, which is an area of active research. Another option is the use of ensemble methods. An ensemble of models evolved by GP can provide a certainty value along with its predictions, but this comes at the cost of the understandability and simplicity of a single model.

Another problem with current GP systems is lack of accuracy of the generated solutions on a class of structurally simple test func-

tions. Functions that exhibit phenotypic behavior highly sensitive to constants in their respective genotype are part of this class. In these cases, an exact solution is only found when a genotypic constant is fitted perfectly, which is nearly impossible to achieve by evolutionary search alone. Korns [2011] provides details and possible solutions.[3]

[3] Korns, M. (2011). Accuracy in symbolic regression. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation Series, pages 129–151. Springer, New York

The previous two problems can also be formulated in a positive manner. In the case of regression, the availability of a symbolic regression algorithm that provably finds exact solutions for all deterministic (noise-free) test functions up to a certain genotypic complexity within a feasible time budget would be a significant achievement.

Among the practical impediments to a wider adoption of GP in general and symbolic regression in particular are a perceived lack of accessible implementations and high compute time requirements of GP. RGP and RGP UI are designed to remove the former impediment to some degree by integrating with the highly popular statistics environment R, by offering a simple graphical user interface, and by their free availability as open source software. The compute time budget needed to arrive at results of suitable quality dropped during the sustained regime of Moore's law, benefiting from the intrinsically parallel nature of population-based evolutionary search. Only recently, GP became a practical method for solving real-world symbolic regression and classification problems in an industrial setting, as demonstrated by the availability of second generation commercial GP systems such as Eureqa and DataModeler.

7.4 *Outlook*

Some of the open questions and problems regarding GP listed in Section 7.3 motivate possible extensions to RGP as well as interesting directions for further research. This final section outlines some of these extensions and research ideas.

Statistical Tools for Symbolic Regression Analysis As Korns [2013] correctly notices, symbolic regression aspires to solve the problem of nonlinear regression, which is canonically formalized by the class of GLMs.[4] A GLM is the linear combination of a set of possibly non-linear basis functions. In principle, a GLM can faithfully represent any non-linear formula, while there are practical limits imposed by model complexity. The symbolic regression problem can therefore be reduced to the problem of finding an optimal set of basis functions for a given data set. By recasting the design and implementation of symbolic regression in RGP in terms of GLM, e.g., by making the concept of basis function explicit, RGP would integrate much better with existing and future tools for GLM analysis available inside the R ecosystem.

[4] Korns, M. (2013). Extreme accuracy in symbolic regression. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation Series, pages 1–30. Springer, New York

Highly-Accurate GP Korns [2013] describes a complex symbolic regression algorithm that is able to provide highly accurate solutions to a large and well-defined class of symbolic regression problems. The basic idea of this algorithm stems from the observation that there are regions of genotypic search space that respond well to evolutionary search, while other regions are much less tractable. These regions are separated by partitioning the search space into islands that are searched in parallel by specialized heuristics. Regions that are intractable by evolutionary search are searched exhaustively. The approach has an additional benefit. During manual search space partitioning, algebraic transformations are used to discover partitions that are phenotypic equivalents. By searching only one partition in each equivalence class, search effort can be reduced significantly. This approach shows promising results for symbolic regression problems with a standard function set, as used in the regression case studies presented in this work, and could be transferred to other GP function sets and applications. Furthermore, the discovery of equivalence classes could be automated to some degree by expert systems representing domain knowledge, e.g., computer algebra systems in the case of symbolic regression.

Domain-Specific GP Systems The practical usability of GP systems would be greatly improved by further integration into domain-specific tools, leading to a new generation of domain-specific GP systems. These systems would be specifically tuned towards a fixed class of application problems in search heuristic parameters and default function set by means introduced in Chapter 5. They would be integrated with other software components implementing data acquisition, preprocessing, and result validation, saving users much of the manual setup and configuration work required in general purpose systems. There are already application domains that use specialized in-house GP systems, such as financial time series analysis or chemical plant modeling and control, and it is to be expected that these systems will become more specialized to their respective application domains in the future.

A
Additional Figures

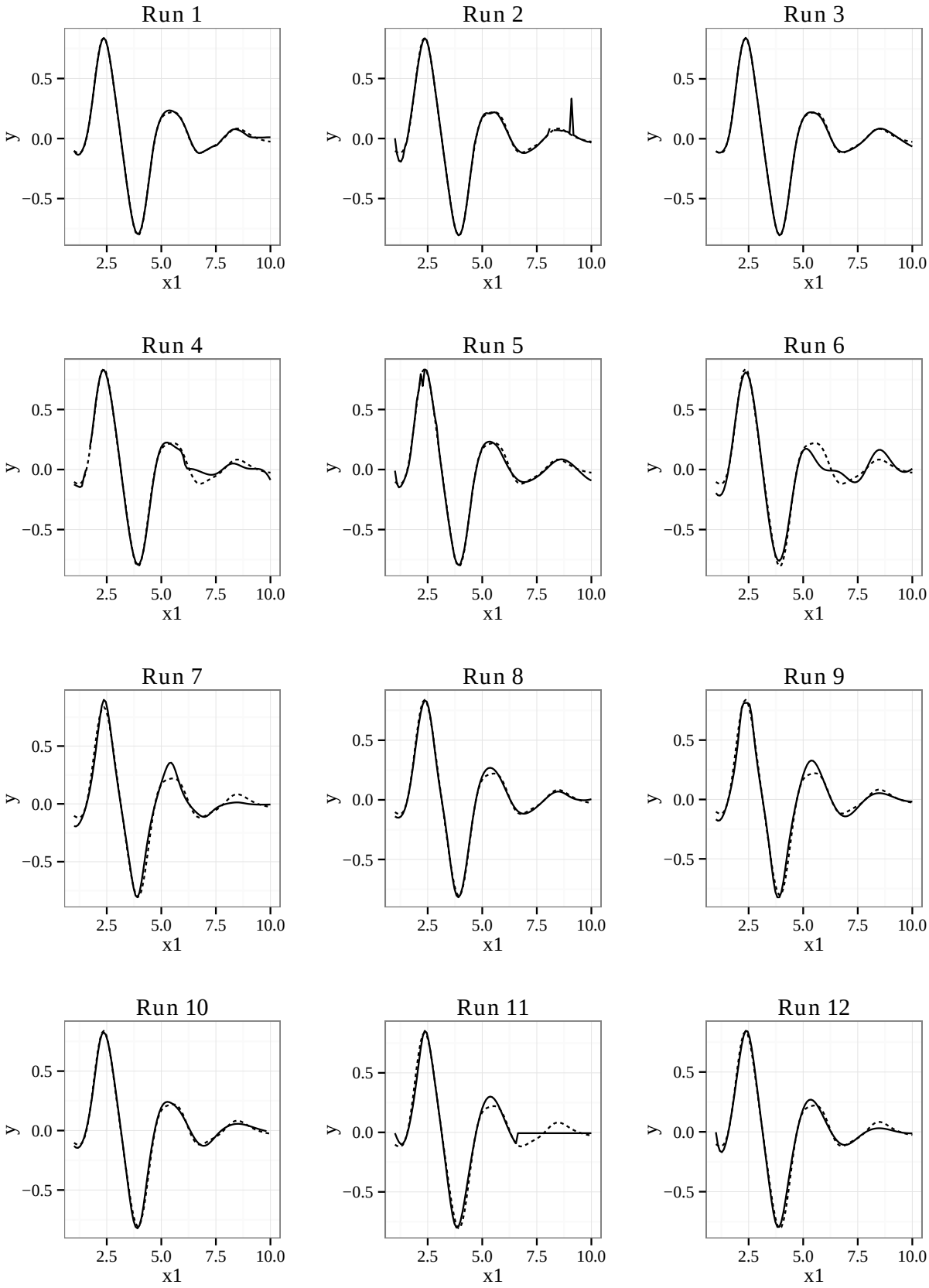


Figure A.1: Plots of the best individual for 12 GMOGP calibration runs (solid line), overlayed by true function (dotted line), as generated by GMOGP-FCA with default settings and a budget of 12 hours.

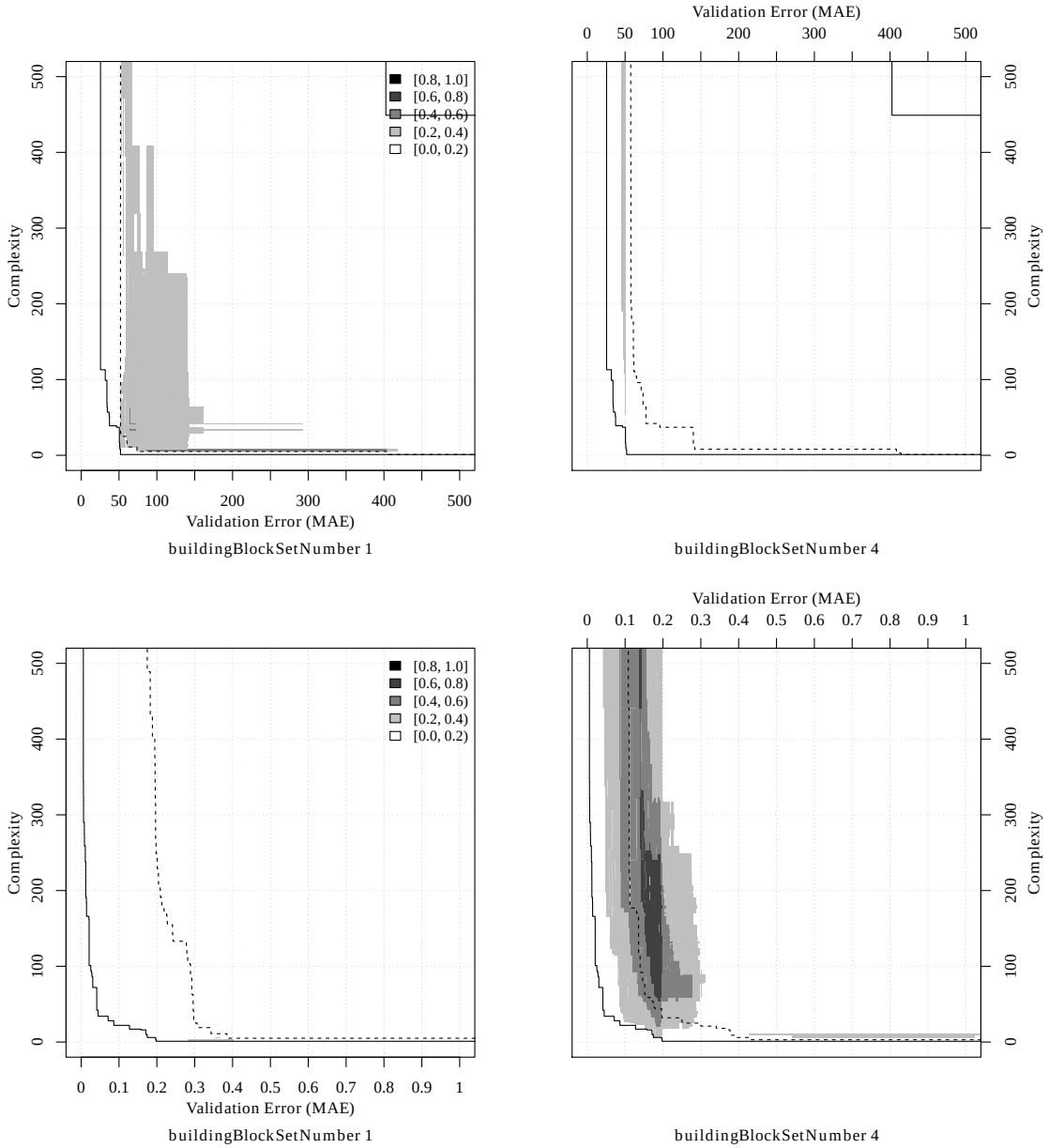


Figure A.2: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P_4, top row) and Salustowicz 1D (P_5, bottom row), i_{funcset} parameter.

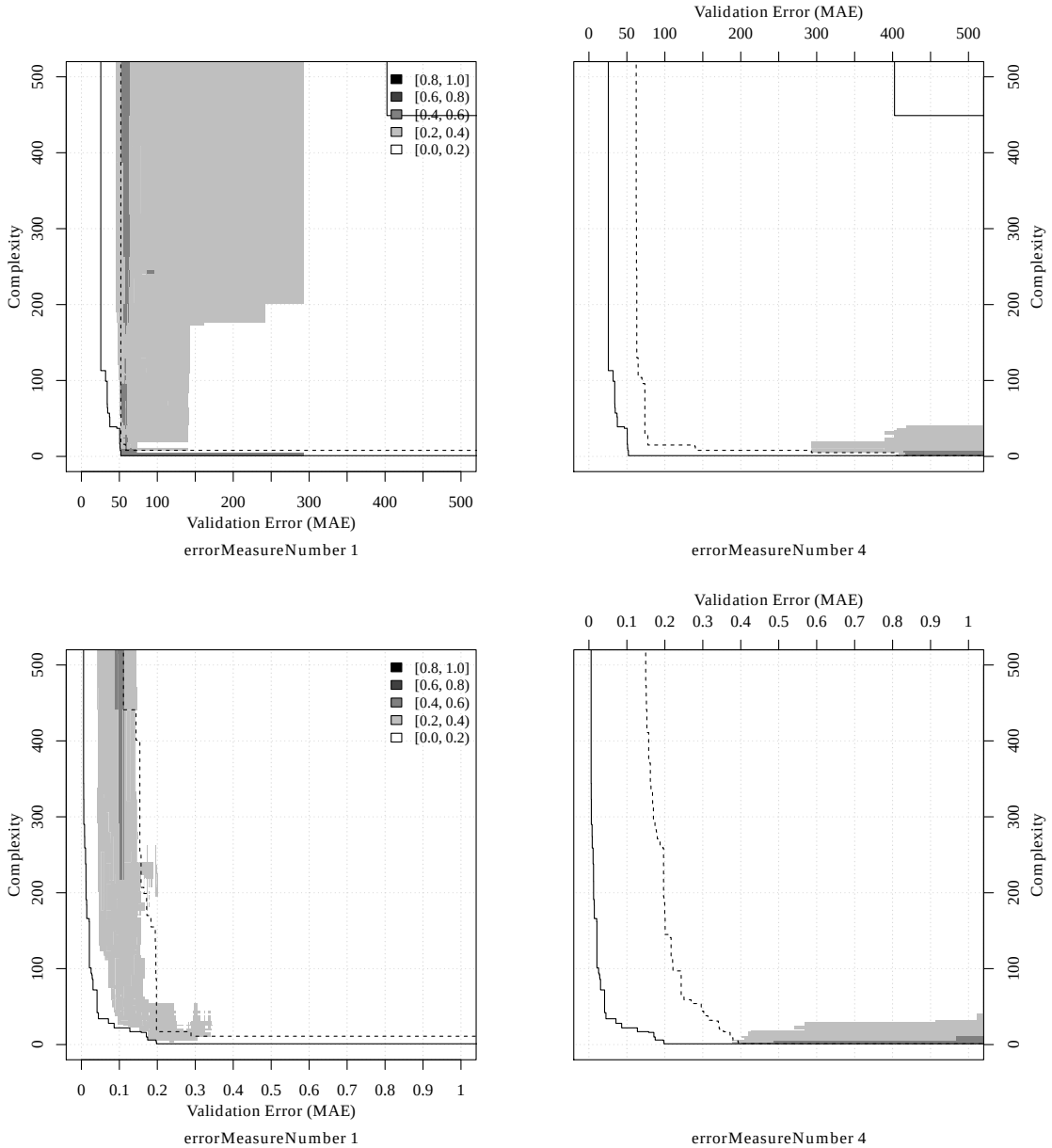


Figure A.3: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P_4, top row) and Salustowicz 1D (P_5, bottom row), i_{error} parameter.

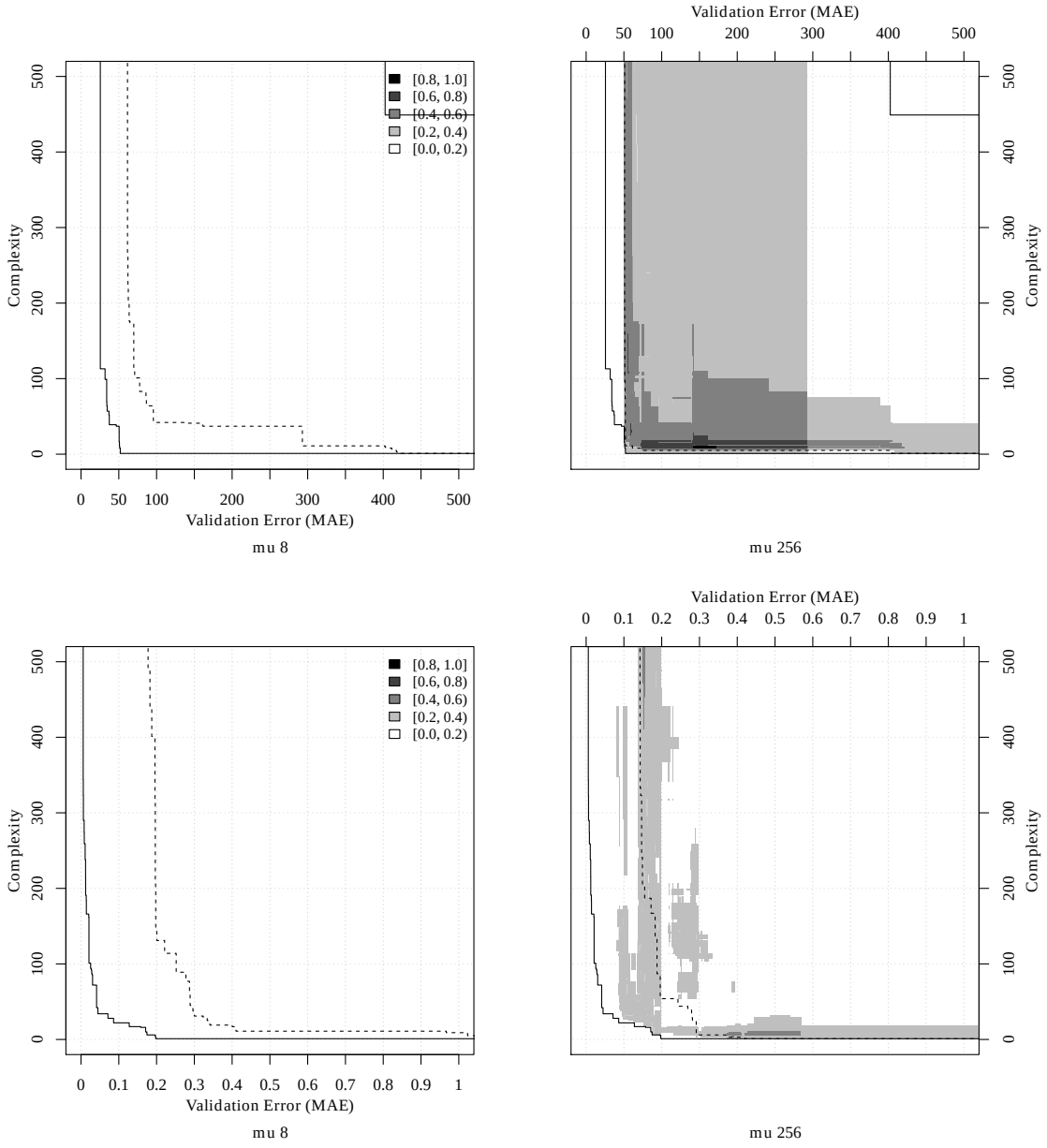


Figure A.4: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P_4, top row) and Salustowicz 1D (P_5, bottom row), μ parameter.

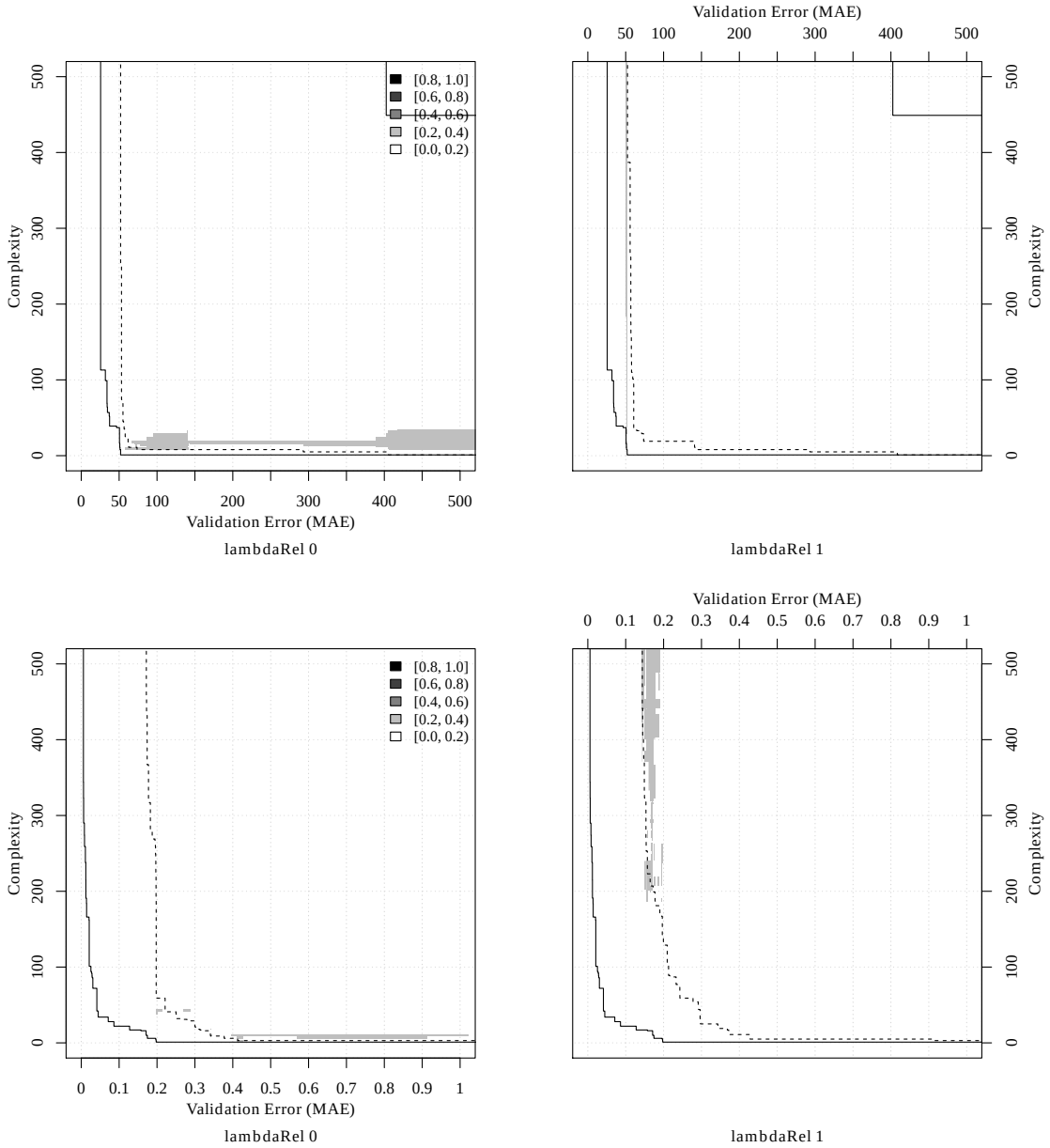


Figure A.5: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P4, top row) and Salustowicz 1D (P5, bottom row), $\lambda_{\mu\,\mathrm{rel}}$ parameter.

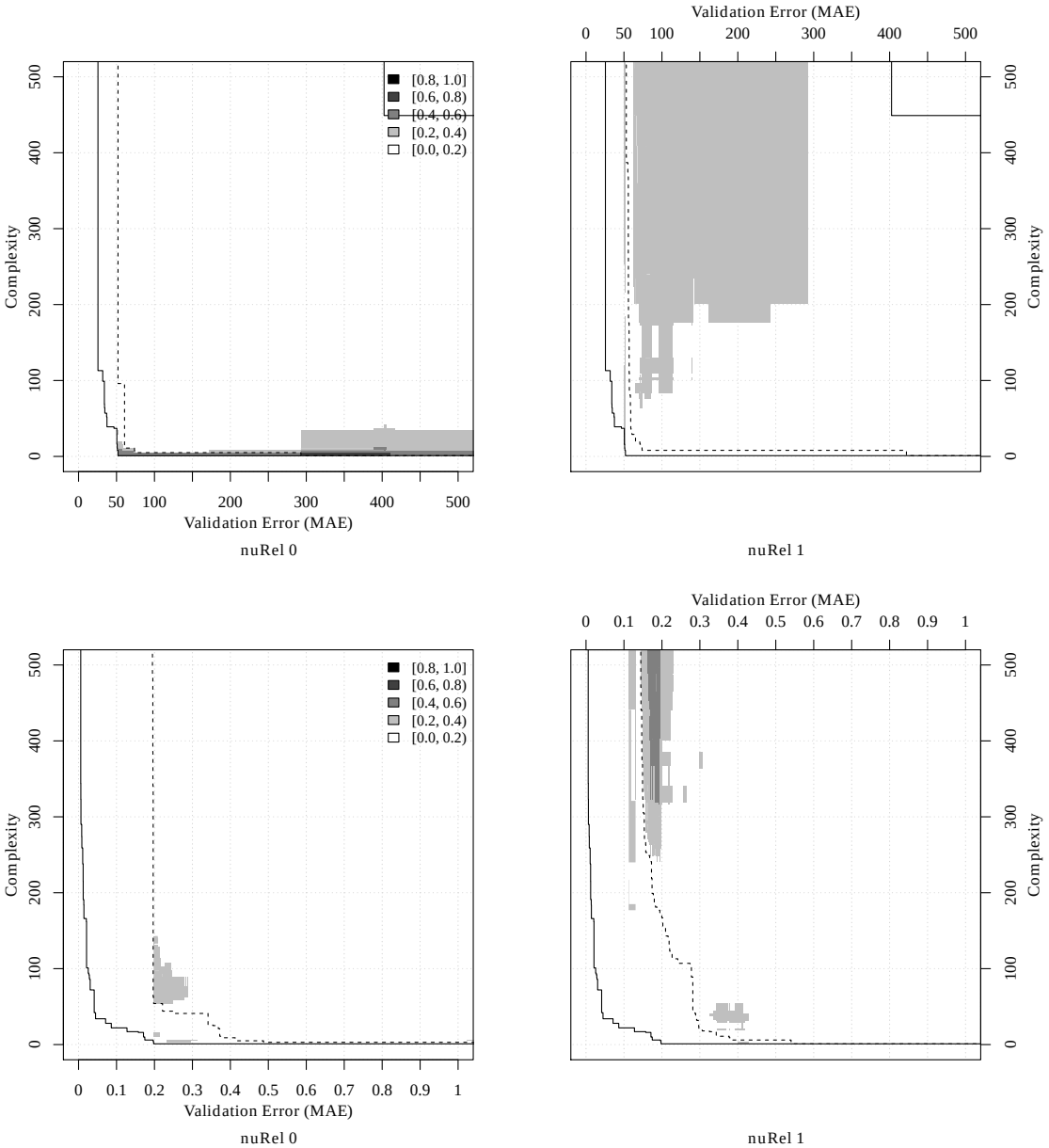


Figure A.6: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P4, top row) and Salustowicz 1D (P5, bottom row), $\nu_{\mu\,\mathrm{rel}}$ parameter.

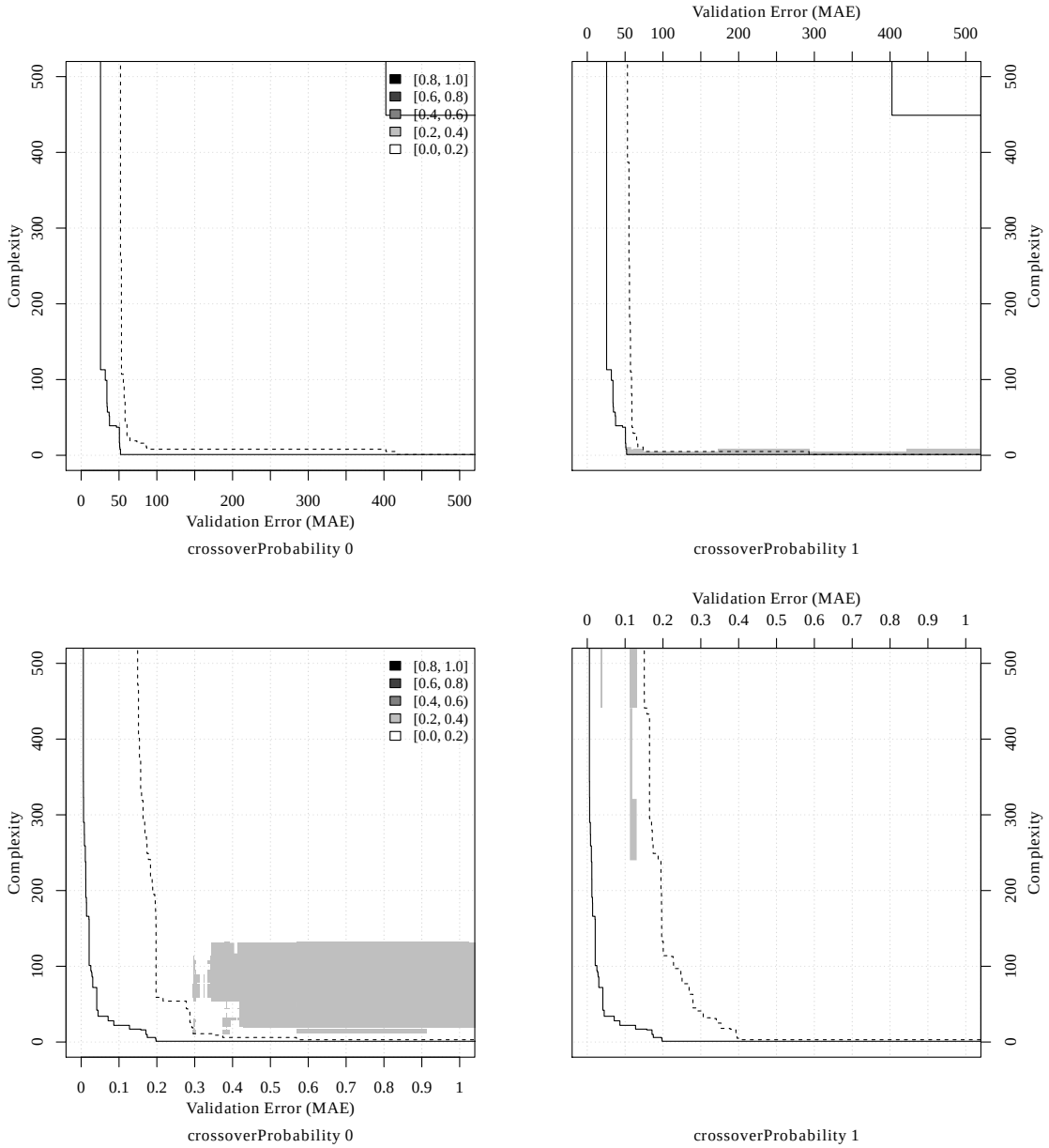


Figure A.7: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P_4, top row) and Salustowicz 1D (P_5, bottom row), $p_{\text{crossover}}$ parameter.

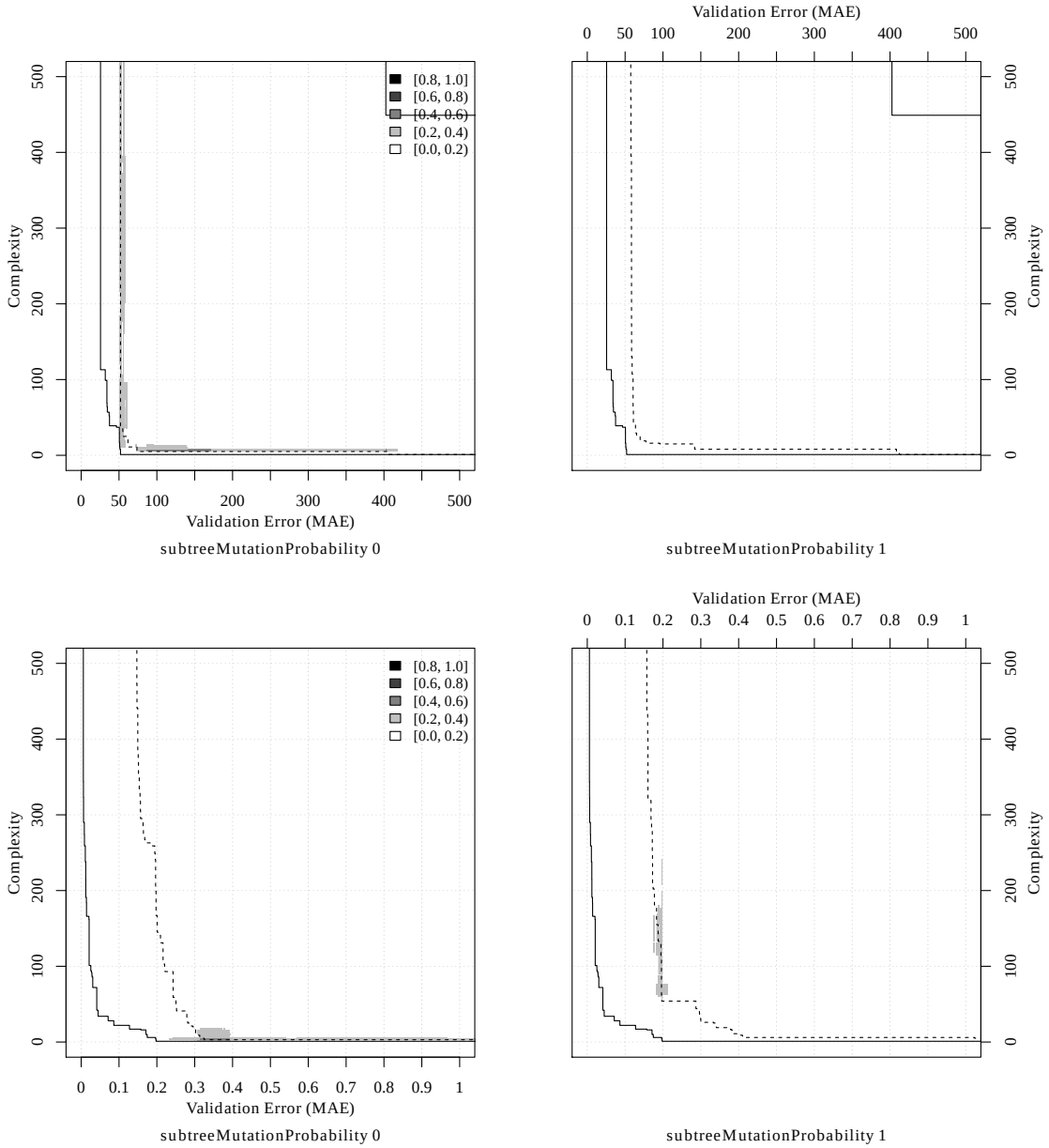


Figure A.8: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P_4, top row) and Salustowicz 1D (P_5, bottom row), p_{msubtree} parameter.

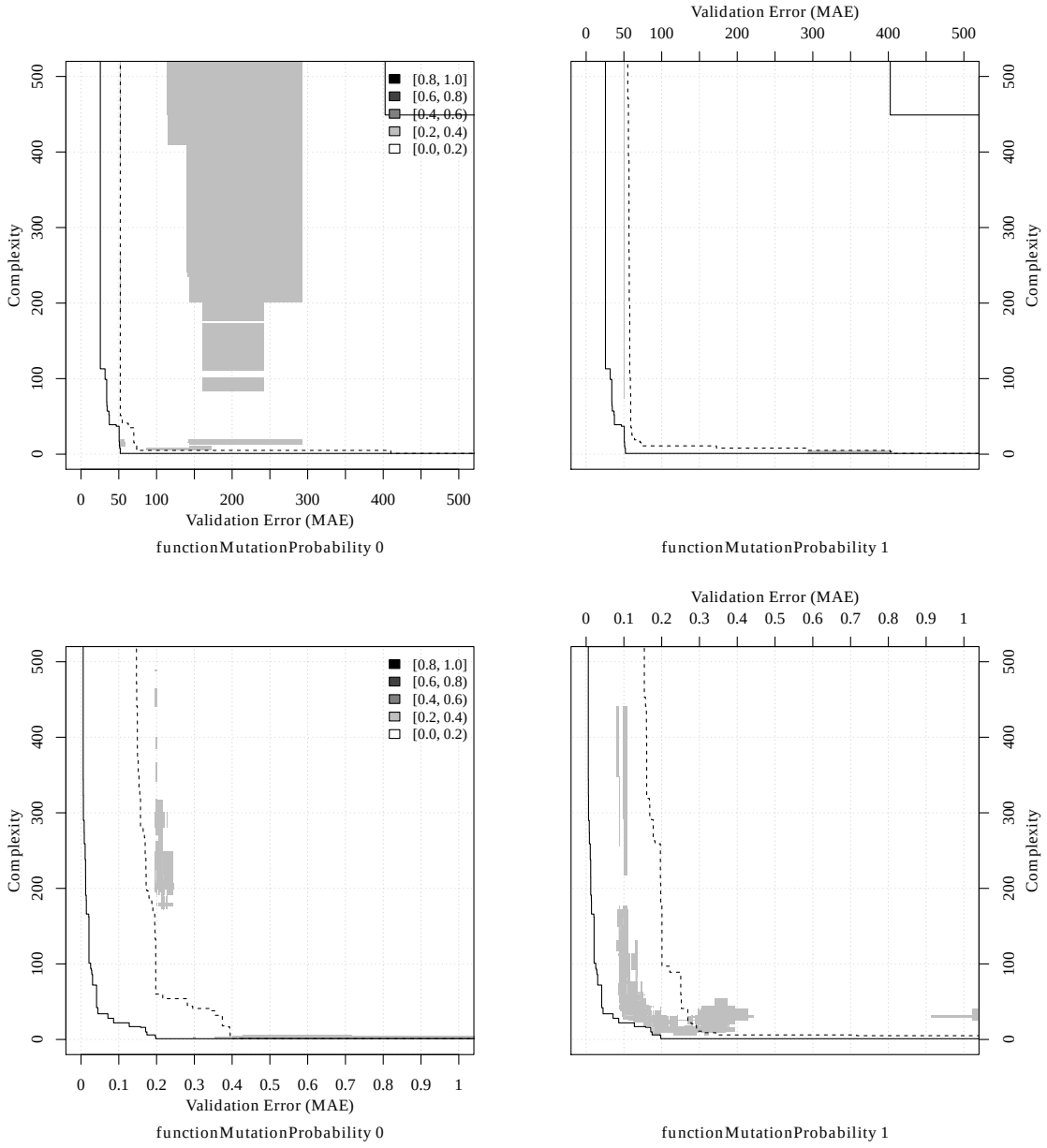


Figure A.9: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P4, top row) and Salustowicz 1D (P5, bottom row), p_{mfunc} parameter.

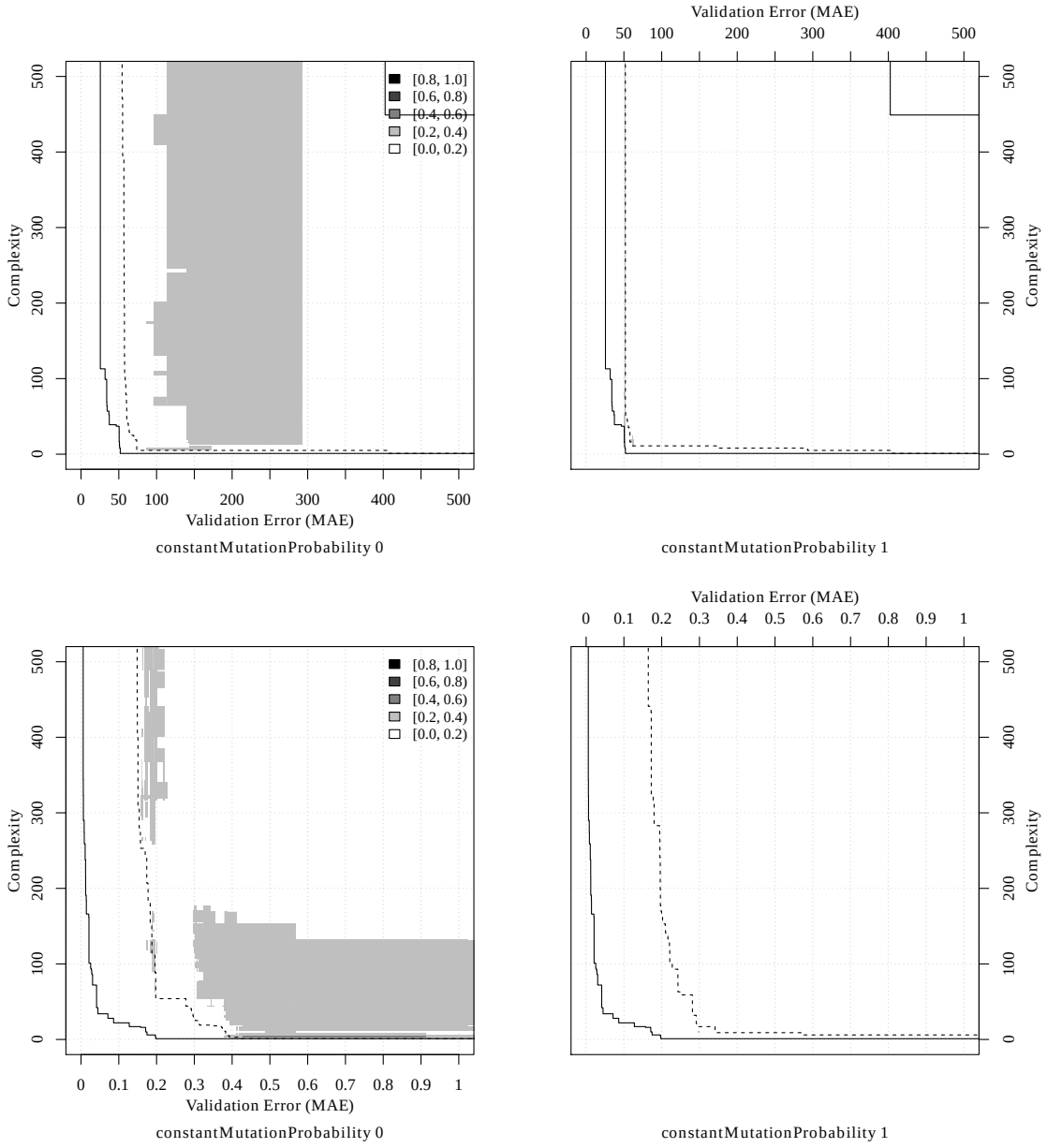


Figure A.10: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P4, top row) and Salustowicz 1D (P5, bottom row), p_{mconst} parameter.

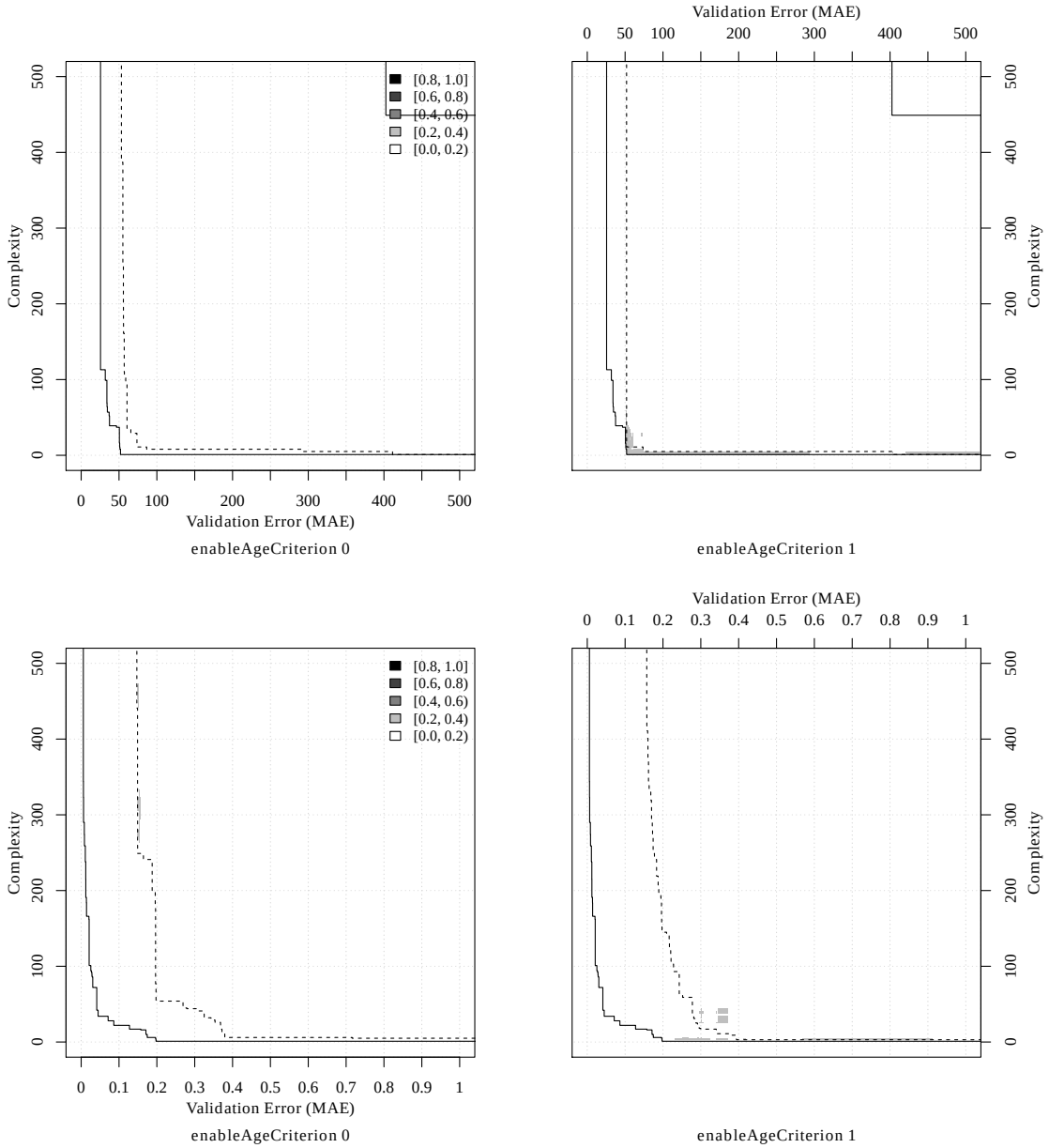


Figure A.11: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P_4, top row) and Salustowicz 1D (P_5, bottom row), b_{age} parameter.

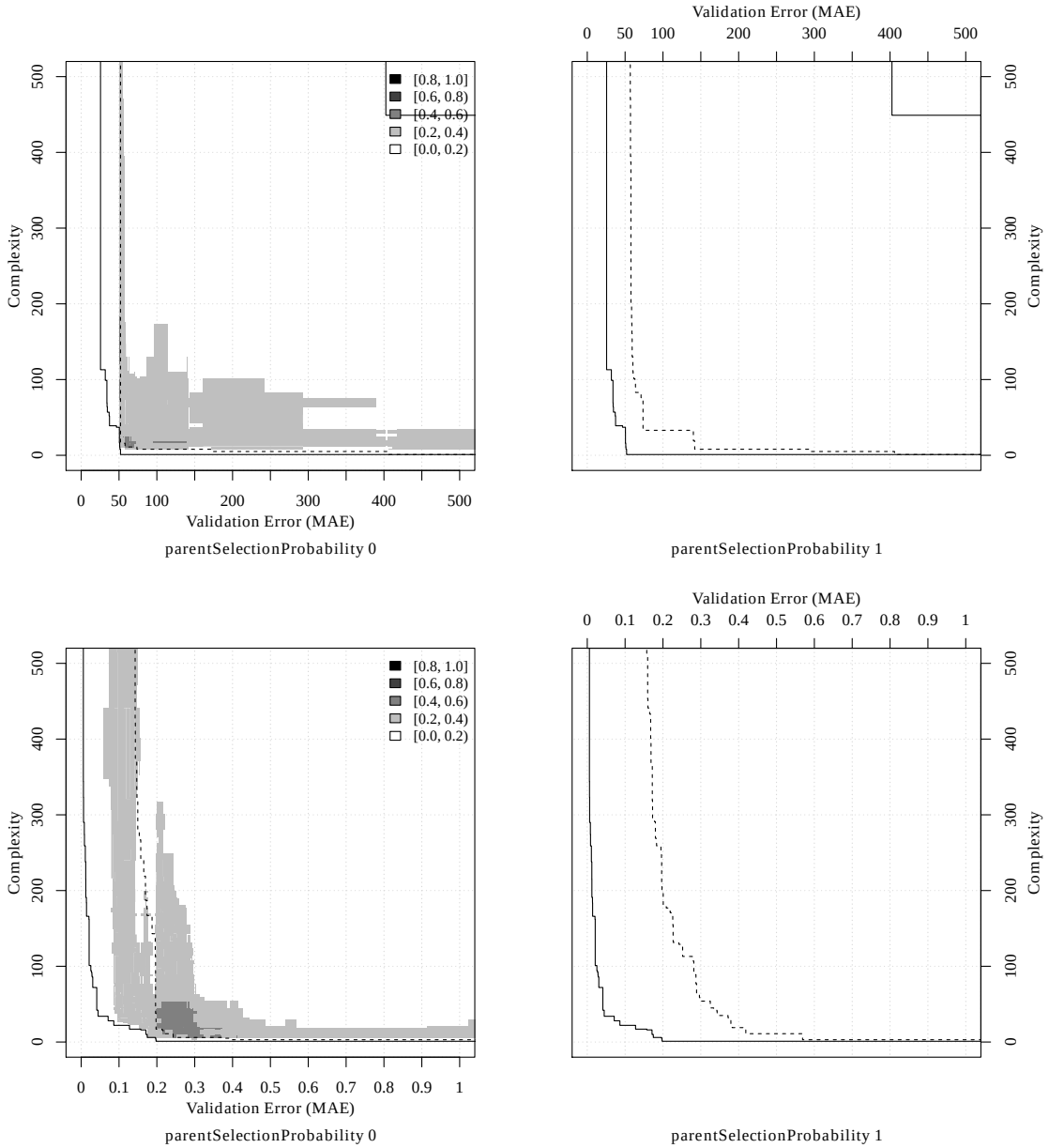


Figure A.12: EAF difference plots for GMOGP-FCA on Air Passengers 1D (P4, top row) and Salustowicz 1D (P5, bottom row), p_{psel} parameter.

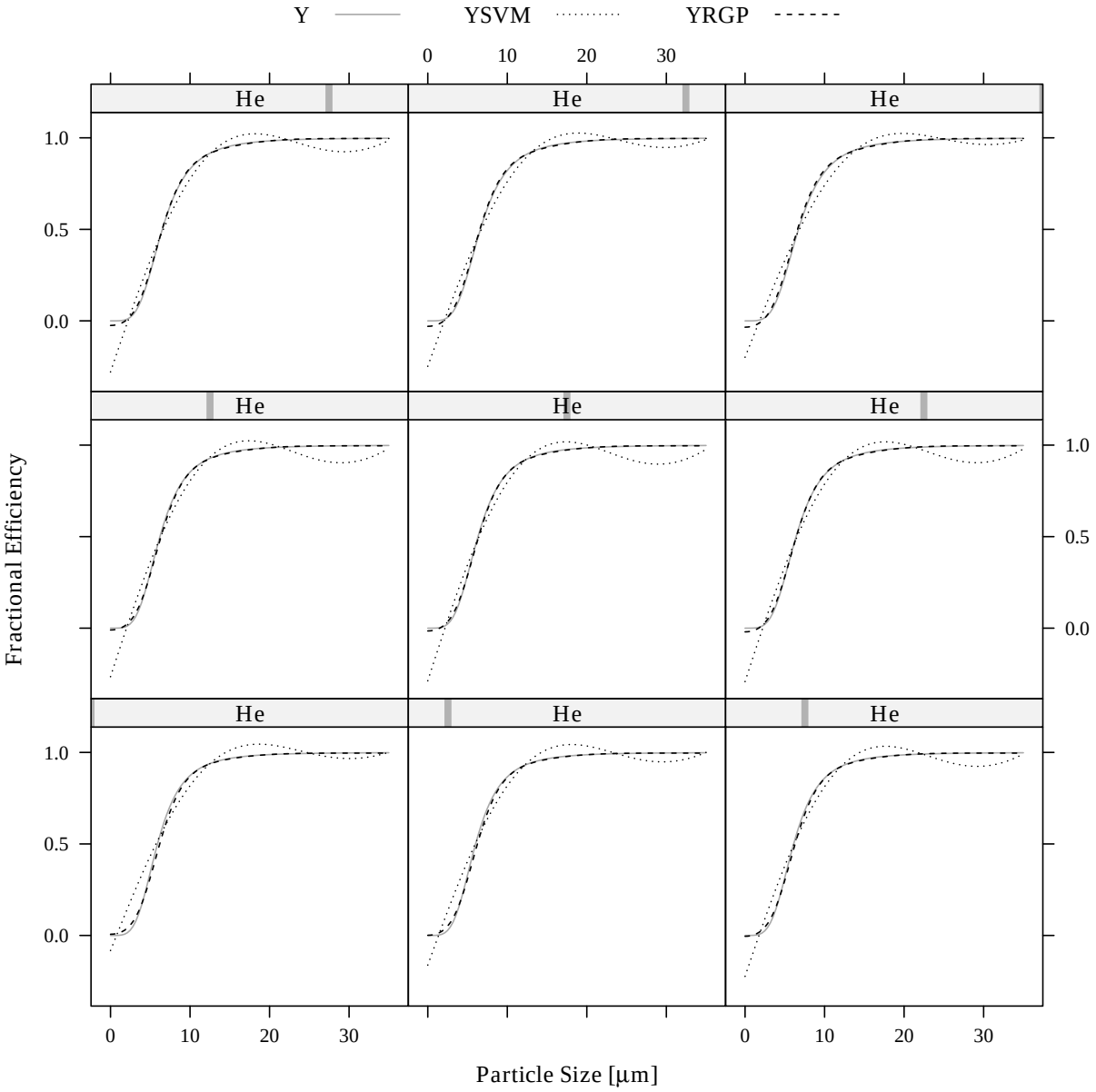


Figure A.13: Cyclone fractional efficiency when varying inlet height (H_e) as predicted by RGP (dashed black line) and SVM (dotted black line) models, compared with true fractional efficiency (solid gray line).

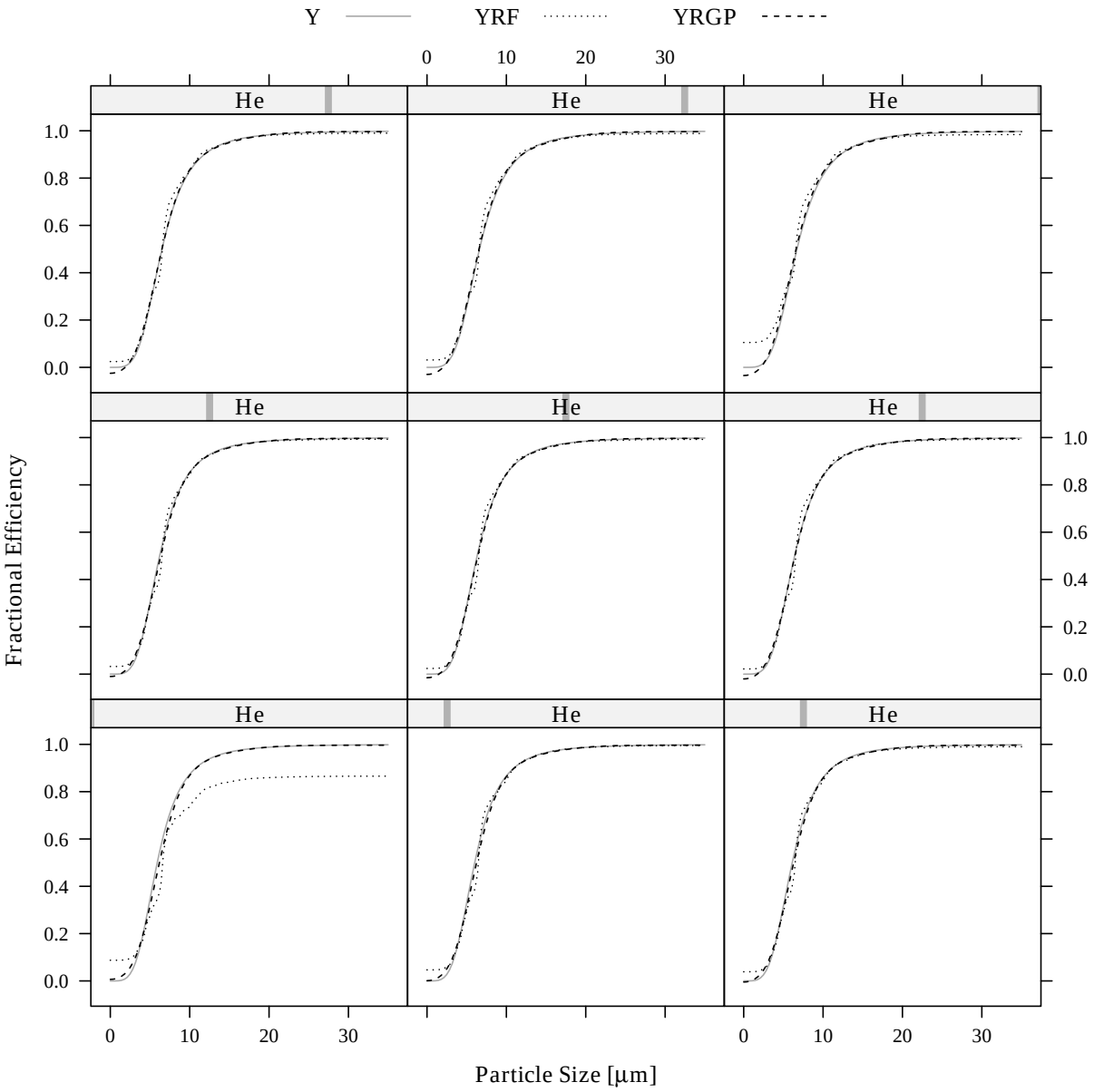


Figure A.14: Cyclone fractional efficiency when varying inlet height (H_e) as predicted by RGP (dashed black line) and RF (dotted black line) models, compared with true fractional efficiency (solid gray line).

About the Author

Studies of computer science and biology at TU Dortmund University and Ruhr-Universität Bochum, Degree: Diplom-Informatiker (10/2000 – 09/2007). Software developer at Dortmund Intelligence Project GmbH, design and implementation of GP systems for financial applications (01/2007 – 05/2009). Research associate at Cologne University of Applied Sciences, research in GP algorithms and systems for financial and technical applications (since 06/2009). Co-founder of sourcewerk UG (haftungsbeschränkt), consultant for simulation software and high-performance computing (since 01/2011).

Publications

O. Flasch, A. Kaspari, K. Morik, and M. Wurst. Aspect-based tagging for collaborative media organization. In B. Berendt et al., editors, *Workshop on Web Mining, WebMine 2006. Revised Selected and Invited Papers*, volume 4737 of *Lecture Notes in Computer Science*, pages 122–141, Berlin, 2007. Springer. ISBN 978-3-540-74950-9.

J. Ziegenhirt, T. Bartz-Beielstein, O. Flasch, W. Konen, and M. Zaefferer. Optimization of biogas production with computational intelligence a comparative study. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1–8, Piscataway, NJ, 2010. IEEE Press.

O. Flasch, T. Bartz-Beielstein, A. Davtyan, P. Koch, W. Konen, T. D. Oyetoyan, and M. Tamutan. Comparing SPO-tuned GP and NARX prediction models for stormwater tank fill level prediction. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1–8, Piscataway, NJ, 2010a. IEEE Press.

O. Flasch, O. Mersmann, and T. Bartz-Beielstein. RGP: An open source genetic programming system for the R environment. In M. Pelikan and J. Branke, editors, *Proceedings of the 12th annual conference on Genetic and evolutionary computation (GECCO 2010)*, pages 2071–2072, New York, 2010b. ACM Press. ISBN 978-1-4503-0073-5.

T. Bartz-Beielstein, M. Friese, M. Zaefferer, B. Naujoks, O. Flasch, W. Konen, and P. Koch. Noisy optimization with sequential

parameter optimization and optimal computational budget allocation. In N. Krasnogor and P. L. Lanzi, editors, *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation (GECCO 2011)*, pages 119–120, New York, 2011. ACM Press. ISBN 978-1-4503-0557-0.

W. Konen, P. Koch, O. Flasch, T. Bartz-Beielstein, M. Friese, and B. Naujoks. Tuned data mining: A benchmark study on different tuners. In N. Krasnogor and P. L. Lanzi, editors, *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation (GECCO 2011)*, pages 1995–2002, New York, 2011. ACM Press. ISBN 978-1-4503-0557-0.

T. Bartz-Beielstein, O. Flasch, and M. Zaefferer. Sequential parameter optimization for symbolic regression. In T. Soule and J. H. Moore, editors, *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation (GECCO 2012)*, pages 495–496, New York, 2012. ACM Press. ISBN 978-1-4503-1178-6.

O. Flasch and T. Bartz-Beielstein. A framework for the empirical analysis of genetic programming system performance. In R. Riolo et al., editors, *Genetic Programming Theory and Practice X*, Genetic and Evolutionary Computation Series, chapter 11, pages 155–169. Springer, New York, 2012.

M. Zaefferer, T. Bartz-Beielstein, M. Friese, B. Naujoks, and O. Flasch. Multi-criteria optimization for hard problems under limited budgets. In T. Soule and J. H. Moore, editors, *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation (GECCO 2012)*, pages 1451–1452, New York, 2012. ACM Press. ISBN 978-1-4503-1178-6.

P. Koch, B. Bischl, O. Flasch, T. Bartz-Beielstein, C. Weihs, and W. Konen. Tuning and evolution of support vector kernels. *Evolutionary Intelligence*, 5(3):153–170, 2012. DOI: 10.1007/s12065-012-0073-8.

O. Flasch, M. Friese, K. Vladislavleva, T. Bartz-Beielstein, O. Mersmann, B. Naujoks, J. Stork, and M. Zaefferer. Comparing ensemble-based forecasting methods for smart-metering data. In A. I. Esparcia-Alcázar, editor, *Applications of Evolutionary Computation - 16th European Conference (EvoApplications 2013)*, volume 7835 of *Lecture Notes in Computer Science*, pages 172–181, Berlin, 2013. Springer. ISBN 978-3-642-37191-2. DOI: 10.1007/978-3-642-37192-9.

M. Zaefferer, B. Breiderhoff, B. Naujoks, M. Friese, J. Stork, A. Fischbach, O. Flasch, and T. Bartz-Beielstein. Tuning multi-objective optimization algorithms for cyclone dust separators. In D. V. Arnold, editor, *Proceedings of the 2014 conference on Genetic and evolutionary computation (GECCO 2014)*, pages 1223–1230, New York, 2014. ACM Press. ISBN 978-1-4503-2662-9.

O. Flasch, M. Friese, M. Zaefferer, T. Bartz-Beielstein, and J. Branke. Learning model-ensemble policies with genetic programming. Cologne Open Science, Schriftenreihe CI-plus TR 03/2015, Bibliothek der Fachhochschule Köln, Betzdorfer Str. 2, 50679 Köln, Germany, 2015.

Index of Abbreviations

AFPO	age-fitness Pareto optimization
ANOVA	analysis of variance
CD	crowding distance
CE	collection efficiency
CFD	computational fluid dynamics
CI	computational intelligence
CRAN	comprehensive R archive network
CSV	comma-separated values
DSL	domain-specific language
EA	evolutionary algorithm
EAF	empirical attainment function
EC	evolutionary computing
EDA	exploratory data analysis
EMOA	evolutionary multi-objective optimization algorithm
ES	evolution strategy
FEM	finite element method
GA	genetic algorithm
GLM	generalized linear model
GMOGP	generational multi-objective genetic programming
GMOGP-F	GMOGP with fitness criterion
GMOGP-FA	GMOGP with fitness and age criteria
GMOGP-FC	GMOGP with fitness and complexity criteria
GMOGP-FCA	GMOGP with fitness, age, and complexity criteria
GP	genetic programming
GSOGP	generational single-objective genetic programming

KM	Kriging model
LM	linear model
MAE	mean-absolute error
MARS	multivariate adaptive regression splines
MSE	mean-square error
NDS	non-dominated sorting
OPGP	ordinal Pareto genetic programming
PCA	principal component analysis
PCR	principal component regression
PL	pressure loss
REPL	read-eval-print loop
RF	random forest
RLD	run-length distribution
RMSE	root-mean-square error
SD	standard deviation
SMSE	scaled-mean-square error
SPO	sequential parameter optimization
SPOT	sequential parameter optimization toolbox
SSE	sum-square error
SVM	support vector machine
XML	extensible markup language

Bibliography

Adler, J. (2010). *R in a nutshell*. O'Reilly, Sebastopol, CA.

Barendregt, H., Abramsky, S., Gabbay, D. M., Maibaum, T. S. E., and Barendregt, H. P. (1992). Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309, New York. Oxford University Press.

Barnes, N. et al. (2013). Science code manifesto. http://sciencecodemanifesto.org/ (retrieved 20.02.2015).

Barth, W. (1956). Berechnung und Auslegung von Zyklonabscheidern auf Grund neuerer Untersuchungen. *Brennstoff-Wärme-Kraft*, 8(1):1–9.

Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation—The New Experimentalism*. Natural Computing Series. Springer, Berlin.

Bartz-Beielstein, T. (2009). Sequential parameter optimization. In Branke, J. et al., editors, *Sampling-based Optimization in the Presence of Uncertainty*, number 09181 in Dagstuhl Seminar Proceedings. Leibniz-Zentrum für Informatik, Germany.

Bartz-Beielstein, T., Branke, J., Mehnen, J., and Mersmann, O. (2014). Evolutionary algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(3):178–195.

Bartz-Beielstein, T., Lasarczyk, C., and Preuss, M. (2005). Sequential parameter optimization. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 773–780, Piscataway, NJ. IEEE Press.

Bartz-Beielstein, T. and Zaefferer, M. (2012). A gentle introduction to sequential parameter optimization. Cologne Open Science, Schriftenreihe CI-plus TR 01/2012, Bibliothek der Fachhochschule Köln, Betzdorfer Str. 2, 50679 Köln, Germany.

Beume, N., Naujoks, B., and Emmerich, M. (2007). SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2009). F-race and iterated F-race: An overview. In Bartz-Beielstein, T.

et al., editors, *Empirical Methods for the Analysis of Optimization Algorithms*. Springer, Berlin.

Bohm, W. and Geyer-Schulz, A. (1996). Exact uniform initialization for genetic programming. In Belew, R. K. and Vose, M., editors, *Foundations of Genetic Algorithms IV*, pages 379–407, San Francisco, CA. Morgan Kaufmann.

Boric, N. and Estevez, P. A. (2007). Genetic programming-based clustering using an information theoretic fitness measure. In Srinivasan, D. and Wang, L., editors, *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 31–38, Piscataway, NJ. IEEE Press.

Box, G., Jenkins, G., and Reinsel, G. (2008). *Time-Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, Hoboken, NJ.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Cortez, P. (2014). *Modern optimization with R*. Use R! Series. Springer International Publishing, Cham, Switzerland.

Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In Grefenstette, J. J., editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, New York. Psychology Press.

De Jong, K. A. (2006). *Evolutionary computation - A unified approach*. MIT Press, Cambridge, MA.

De Jong, K. A. (2007). Parameter setting in EAs: a 30 year perspective. In Lobo, F. G. et al., editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 1–18. Springer, Berlin.

De Lima, E. B., Pappa, G. L., de Almeida, J. M., Goncalves, M. A., and Meira, W. (2010). Tuning genetic programming parameters with factorial designs. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010)*, Piscataway, NJ. IEEE Press.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Droste, S., Jansen, T., and Wegener, I. (2000). Optimization with randomized search heuristics: The (A)NFL theorem, realistic scenarios, and difficult functions. Interner Bericht des Sonderforschungsbereichs 531 *Computational Intelligence* CI–91/00, Universität Dortmund, Germany.

Droste, S. and Wiesmann, D. (2000). Metric-based evolutionary algorithms. In *Proceedings of the European Conference on Genetic Programming (EuroGP 2000)*, volume 1802 of *Lecture Notes in Computer Science*, pages 29–43, Berlin. Springer.

Drucker, H., Kaufman, B. L., Smola, A., and Vapnik, V. (1997). Support vector regression machines. In *Advances in Neural Information Processing Systems*, volume 9, pages 155–161.

Ebert, T., Belz, J., and Nelles, O. (2014). Detektion von Extrapolation. *Proceedings of the 24. Workshop on Computational Intelligence, Dortmund, Germany*, 50:17–32.

Eiben, A., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141.

Espejo, P. G., Ventura, S., and Herrera, F. (2010). A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(2):121–144.

Feldt, R. and Nordin, P. (2000). Using factorial experiments to evaluate the effect of genetic programming parameters. In Poli, R. et al., editors, *Proceedings of the European Conference on Genetic Programming (EuroGP 2000)*, volume 1802 of *Lecture Notes in Computer Science*, pages 271–282, Berlin. Springer.

Flajolet, P. and Sedgewick, R. (2009). *Analytic Combinatorics*. Cambridge University Press, New York, 1st edition.

Flasch, O. and Bartz-Beielstein, T. (2012). A framework for the empirical analysis of genetic programming system performance. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice X*, Genetic and Evolutionary Computation Series. Springer, New York.

Forrester, A., Sobester, A., and Keane, A. (2008). *Engineering Design via Surrogate Modelling*. Wiley, Chichester, U.K., 1st edition.

Forsyth, R. S. (1986). Evolutionary learning strategies. In Forsyth, R. S. and Rada, R., editors, *Machine Learning Applications in Expert Systems and Information Retrieval*, pages 78–95, Chichester, U.K. Ellis Horwood.

Friedman, J. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67.

Friese, M., Stork, J., Guerra, R. R., Bartz-Beielstein, T., Thaker, S., Flasch, O., and Zaefferer, M. (2013). UniFIeD: Univariate frequency-based imputation for time series data. Cologne Open Science, Schriftenreihe CI-plus TR 05/2013, Bibliothek der Fachhochschule Köln, Betzdorfer Str. 2, 50679 Köln, Germany.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading, MA.

Grönmping, U. (2014). R package FrF2 for creating and analyzing fractional factorial 2-level designs. *Journal of Statistical Software*, 56(1):1–56.

Hansen, N., Finck, S., and Ros, R. (2011). Coco - Comparing continuous optimizers: The documentation. Technical Report RT-0409, INRIA, France.

Hastie, T. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer, New York, 2nd edition.

Hornby, G. S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In Keijzer, M. et al., editors, *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO 2006)*, volume 1, pages 815–822, Seattle, WA. ACM Press.

Hughes, J. (1989). Why functional programming matters. *The Computer Journal*, 32(2):98–107.

Hyndman, R. J. (2006). Another look at forecast accuracy metrics for intermittent demand. *Foresight: The International Journal of Applied Forecasting*, 2006(4):43–46.

Iba, H. (1996). Random tree generation for genetic programming. In Voigt, H. et al., editors, *Parallel Problem Solving from Nature IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 144–153, Berlin. Springer.

Jolliffe, I. (1982). A note on the use of principal components in regression. *Applied Statistics*, 31(3):300–303.

Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA. Morgan Kaufmann.

Keijzer, M. (2004). Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):59–269.

Keijzer, M. and Babovic, V. (2002). Declarative and preferential bias in GP-based scientific discovery. *Genetic Programming and Evolvable Machines*, 3(1):41–79.

Keijzer, M. and Foster, J. (2007). Crossover bias in genetic programming. In Ebner, M. et al., editors, *Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 33–44. Springer, Berlin.

Kieseppä, I. A. (1997). Akaike information criterion, curve-fitting and the philosophical problem of simplicity. *British Journal for the Philosophy of Science*, 48(1):21–48.

Kitzelmann, E. (2010). Inductive programming: A survey of program synthesis techniques. In Schmid, U. et al., editors, *Approaches and Applications of Inductive Programming*, volume 5812 of *Lecture Notes in Computer Science*, pages 50–73. Springer, Berlin.

Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., and Konen, W. (2011). On the tuning and evolution of support vector kernels. Cologne Open Science, CIOP Technical Report 04/11, Bibliothek der Fachhochschule Köln, Betzdorfer Str. 2, 50679 Köln, Germany.

Korns, M. (2011). Accuracy in symbolic regression. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation Series, pages 129–151. Springer, New York.

Korns, M. (2013). Extreme accuracy in symbolic regression. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation Series, pages 1–30. Springer, New York.

Kotanchek, M., Smits, G., and Vladislavleva, E. (2006). Pursuing the Pareto paradigm: Tournaments, algorithm variations & ordinal optimization. In Riolo, R. L. et al., editors, *Genetic Programming Theory and Practice IV*, Genetic and Evolutionary Computation Series, pages 167–186. Springer, New York.

Kotanchek, M. E., Vladislavleva, E., and Smits, G. F. (2010). Symbolic regression via genetic programming as a discovery engine: Insights on outliers and prototypes. In Riolo, R. et al., editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation Series, pages 55–72. Springer, New York.

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

Koza, J. R. (1994). *Genetic programming II: Automatic discovery of reusable programs*. MIT Press, Cambridge, MA.

Koza, J. R. and Bennett III, F. H. (1999). Automatic synthesis, placement, and routing of electrical circuits by means of genetic programming. In Spector, L. et al., editors, *Advances in Genetic Programming 3*, pages 105–134. MIT Press, Cambridge, MA.

Kruskal, W. and Wallis, W. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621.

Lasarczyk, C. W. G. (2007). Genetische Programmierung einer algorithmischen Chemie. Dissertation, TU Dortmund, Germany.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomForest. *R News*, 2(3):18–22. R package version 4.6.10, http://CRAN.R-project.org/doc/Rnews/ (retrieved 20.02.2015).

López-Ibáñez, M., Paquete, L., and Stützle, T. (2010). Exploratory analysis of stochastic local search algorithms in biobjective optimization. In Bartz-Beielstein, T. et al., editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 209–222. Springer, Berlin.

Luke, S. (2013). The ECJ owner's manual – A user manual for the ECJ evolutionary computation library. 21th edition, http://cs.gmu.edu/ eclab/projects/ecj/docs/manual/manual.pdf (retrieved 20.02.2015).

Luke, S. and Panait, L. (2002). Lexicographic parsimony pressure. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 829–836, San Francisco, CA. Morgan Kaufmann.

Luke, S. and Spector, L. (1997). A comparison of crossover and mutation in genetic programming. In Koza, J. R. et al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 240–248, San Francisco, CA. Morgan Kaufmann.

Mayo, D. G. (1996). *Error and the Growth of Experimental Knowledge*. The University of Chicago Press, Chicago, IL.

Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., and Rudolph, G. (2011). Exploratory landscape analysis. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation (GECCO 2011)*, pages 829–836, New York. ACM Press.

Mevik, B., Wehrens, R., and Liland, K. H. (2013). *pls: Partial Least Squares and Principal Component regression*. R package version 2.4-3, http://CRAN.R-project.org/package=pls (retrieved 20.02.2015).

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2014). *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.6-4, http://CRAN.R-project.org/package=e1071 (retrieved 20.02.2015).

Milborrow, S. (2014). *earth: Multivariate Adaptive Regression Spline Models*. R package version 3.2-7, http://CRAN.R-project.org/package=earth (retrieved 20.02.2015).

Moraglio, A., Krawiec, K., and Johnson, C. G. (2012). Geometric semantic genetic programming. In *Parallel Problem Solving from Nature-PPSN XII*, pages 21–31. Springer, Berlin.

Morandat, F., Hill, B., Osvald, L., and Vitek, J. (2012). Evaluating the design of the R language - Objects and functions for data analysis. In Noble, J., editor, *ECOOP*, volume 7313 of *Lecture Notes in Computer Science*, pages 104–131, Berlin. Springer.

Nordin, P. and Banzhaf, W. (1997). Genetic reasoning: Evolving proofs with genetic search. In Koza, J. R. et al., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 255–260, San Francisco, CA. Morgan Kaufmann.

Oakland, J. (2003). *Statistical Process Control*. Quality management Series. Butterworth-Heinemann, Newton, MA.

Parkes, A. J. and Walser, J. P. (1996). Tuning local search for satisfiability testing. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI 1996)*, pages 356–362, Menlo Park, CA. AIII Press.

Pierce, B. C. (2002). *Types and Programming Languages*. MIT Press, Cambridge, MA.

Piszcz, A. and Soule, T. (2006a). Genetic programming: Analysis of optimal mutation rates in a problem with varying difficulty. In Sutcliffe, G. C. J. and Goebel, R. G., editors, *Proceedings of the 19th International Florida AI Research Society Conference (FLAIRS 2006)*, pages 451–456, Menlo Park, CA. AIII Press.

Piszcz, A. and Soule, T. (2006b). A survey of mutation techniques in genetic programming. In Keijzer, M. et al., editors, *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO 2006)*, volume 1, pages 951–952, Seattle, WA. ACM Press.

Pohlheim, H. (2006). Multidimensional scaling for evolutionary algorithms - Visualization of the path through search space and solution space using sammon mapping. *Artificial Life*, 12(2):203–209.

Poli, R. (2001). Exact schema theory for genetic programming and variable-length genetic algorithms with one-point crossover. *Genetic Programming and Evolvable Machines*, 2(2):123–163.

Poli, R. and Langdon, W. B. (1998). On the ability to search the space of programs of standard, one-point and uniform crossover in genetic programming. Technical Report CSRP-98-7, School of Computer Science, University of Birmingham, U.K.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. http://lulu.com (retrieved 20.02.2015). (With contributions by J. R. Koza).

Raidl, G. R. and Gottlieb, J. (2005). Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4):441–475.

Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann, Newton, MA, 2nd edition.

Roustant, O., Ginsbourger, D., and Deville, Y. (2012). DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software*, 51(1):1–55. R package version 1.5.3, http://www.jstatsoft.org/v51/i01/ (retrieved 20.02.2015).

RStudio Inc. (2014). *shiny: Web Application Framework for R*. R package version 0.10.1, http://CRAN.R-project.org/package=shiny (retrieved 20.02.2015).

Sacks, J., Welch, W., Mitchell, T., and Wynn, H. (1989). Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423.

Salustowicz, R. and Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141.

Schmidt, M. D. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85.

Schmidt, M. D. and Lipson, H. (2010). Age-fitness Pareto optimization. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation (GECCO 2010)*, pages 543–544, New York. ACM Press.

Schöning, U. and Pruim, R. J. (1998). *Gems of Theoretical Computer Science*. Springer, Berlin.

Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall, Upper Saddle River, NJ, 1st edition.

Shearer, C. (2000). The CRISP-DM model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4):13–22.

Shir, O. M. (2012). Niching in evolutionary algorithms. In Rozenberg, G. et al., editors, *Handbook of Natural Computing*, pages 1035–1069. Springer, Berlin.

Silva, S. (2008). *Controlling bloat: Individual and population based approaches in genetic programming*. PhD thesis, Departamento de Engenharia Informatica, Universidade de Coimbra, Portugal.

Sing, T., Sander, O., Beerenwinkel, N., and Lengauer, T. (2005). ROCR: Visualizing classifier performance in R. *Bioinformatics*, 21(20):3940–3941.

Smits, G. and Kotanchek, M. (2004). Pareto-front exploitation in symbolic regression. In O'Reilly, U. et al., editors, *Genetic Programming Theory and Practice II*, Genetic and Evolutionary Computation Series, pages 283–299. Springer, New York.

Smits, G. and Vladislavleva, E. (2006). Ordinal Pareto genetic programming. In Yen, G. G. et al., editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 3114–3120, Piscataway, NJ. IEEE Press.

Stadler, B. M. R., Stadler, P. F., Wagner, G. P., and Fontana, W. (2000). The topology of the possible: Formal spaces underlying patterns of evolutionary change. Working papers, Santa Fe Institute.

Stadler, P. F. and Reidys, C. M. (2002). Neutrality in fitness landscapes. *Applied Mathematics and Computation*, 117(2–3):321–350.

Sun, Y., Wierstra, D., Schaul, T., and Schmidhuber, J. (2009). Efficient natural evolution strategies. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCo 2009)*, pages 539–546, New York. ACM Press.

Therneau, T. M. and Atkinson, E. J. (1997). An introduction to recursive partitioning using the RPART routines. Technical Report 61, Department of Health Science Research, Mayo Clinic, Rochester NY.

Tukey, J. W. (1977). *Exploratory Data Analysis*. Behavioral Science: Quantitative Methods. Addison-Wesley, Reading, MA.

Vladislavleva, E., Smits, G. F., and Den Hertog, D. (2009). Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349.

Wagoner, R. and Chenot, J. (2005). *Metal Forming Analysis*. Cambridge University Press, New York.

Ward, M. P. (1995). Language oriented programming. *Software – Concepts and Tools*, 15(4):147–161.

White, D. R. and Poulding, S. (2009). A rigorous evaluation of crossover and mutation in genetic programming. In Vanneschi, L. et al., editors, *Proceedings of the 12th European Conference on Genetic Programming (EuroGP 2009)*, volume 5481 of *Lecture Notes in Computer Science*, pages 220–231, Berlin. Springer.

Wiesmann, D. (2001). Anwendungsorientierter Entwurf evolutionärer Algorithmen. Dissertation, Universität Dortmund, Germany.

Willis, M., Hiden, H., Marenbach, P., McKay, B., and Montague, G. A. (1997). Genetic programming: An introduction and survey of applications. In Zalzala, A., editor, *Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA 97)*, pages 314–319, London, U.K. IEE Conference Publications (No. 446).

Willmott, C. J. and Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30(1):79–82.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Xie, Y. (2013). *Dynamic Documents with R and knitr*. Chapman and Hall, Boca Raton, FL.

Zaefferer, M., Breiderhoff, B., Naujoks, B., Friese, M., Stork, J., Fischbach, A., Flasch, O., and Bartz-Beielstein, T. (2014). Tuning multi-objective optimization algorithms for cyclone dust separators. In *Proceedings of the 2014 conference on Genetic and evolutionary computation (GECCO 2014)*, New York. ACM Press.

Zimek, A., Schubert, E., and Kriegel, H. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, 5(5):363–387.

Zongker, D. and Punch, B. (1996). *lil-gp 1.01 User's Manual*. East Lansing, MI. Michigan State University.