AUTOMATIC ONLINE ALGORITHM SELECTION FOR OPTIMIZATION IN CYBER-PHYSICAL PRODUCTION SYSTEMS

Dissertation

zur Erlangung des Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

ANDREAS FISCHBACH

Dortmund

2023

Tag der mündlichen Prüfung:
10. Juli 2023

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

Gutachter:
Prof. Dr. Günter Rudolph, Technische Universität Dortmund
Prof. Dr. Thomas Bartz-Beielstein, Technische Hochschule Köln

Acknowledgments

This thesis would not have been possible without the support of many people. This page is dedicated to them. I would like to thank my supervisor Prof. Dr. Thomas Bartz-Beielstein, who intensified my interest and knowledge in this field of research and always motivated and supported me, and my advisor Prof. Dr. Günter Rudolph. Our discussions always provided new impulses and opened up questions that I might not have touched otherwise. Thanks to my mentor Prof. Dr. Thomas Schwentick, especially for the helpful discussions from an interdisciplinary perspective on the whole process of writing this thesis.

Thanks to my close office mates, Martina, Jörg, Martin, Frederick and Patrick, for deep discussions about certain research topics as well as about daily life. They also made it a pleasant time.

My PhD project is closely related to the KOARCH project. Therefore, I would like to thank my project colleagues, Jan Strohschein and Andreas Bunte, for many helpful and intensive discussions, which were very beneficial for my research and this thesis. This work would not have been possible without the financial support of of the German Federal Ministry of Education and Research in the funding program *Forschung an Fachhochschulen* under the grant number 13FH007IB6.

I would also like to thank Prof. Dietlind Zühlke and Steffen Moritz, our meetings were so motivating, especially in the circumstances of the covid pandemic where almost all of us were working remotely. It was good to keep in touch!

Thanks to Markus Baum and Prof. Denis Anders, the feedback and support I received gave me a huge boost in the final phase of my thesis project.

I would also like to thank all my colleagues at the Innovation Hub, especially Jörn Richter, Jannik Blähser, Ben Rudat, Manuel Hecker, Julian Lengelsen, Daniela Reschke, Cathrin Pape, Tobias Hund, Daniel Hillen, Martin Höffer and Christoph Becker.

Last but not least, I would like to thank my son Julius and my wife Agnes (they mean the world to me) for their love, patience, support and enthusiasm(!) during these years.

Contents

Essentially, all models are wrong, but some are useful
George E. P. Box (1976)

Chapter 1

Introduction

This thesis documents the research and development of an architecture for cyber-physical production systems with a primary focus on the optimization of the control parameters of variable and adaptive production processes. The algorithmic solution developed utilizes process data to generate adequate test functions and test and tune feasible algorithms during runtime for real-world problems. In combination with the application of state-of-the-art methods from benchmarking, this enables an automatic selection of the most promising algorithm from a given portfolio according to established performance criteria. This chapter gives the motivation and introduces the problems and challenges for the aimed solution.

1.1 Motivation

The 4th industrial revolution, named Industry 4.0 (I4.0) by the german government as a strategic initiative in 2011, aims at creating intelligent factories [58, 159, 158]. To cope with challenges like shrinking product life cycles, progress in market penetration of innovative product technologies, and increasing demand for product individualization, a significant increase in the degree of automation of production processes is needed [121].

The desired levels of automation and adaption of production processes can not be reached with conventional manufacturing processes and plants, as high amounts of manual effort would be needed [7] to, i.e., realize changes in the topology of manufacturing plants and optimization of the process itself and the products. The development of systems that integrate computation with physical processes has the potential to solve that problem. Such systems are called Cyber-physical Systems (CPSs). For a Definition of CPS, we cite Monostori [105, p.1] and extend the definition of CPS to Cyber-physical Production System (CPPS):

Definition 1 (Cyber-physical production system)**.** *"Cyber-Physical Systems (CPS) are systems of collaborating computational entities which are in intensive connection with the surrounding physical world and its on-going processes, providing and using, at the same time, data-accessing and data-processing services available on the internet. With other words, CPS can be generally characterized as 'physical and engineered systems whose operations are monitored, controlled, coordinated, and integrated by a computing and communicating core'." [128] CPPS are those CPS, that are realizing a production process.*

Monostori highlights the need to especially understand the interaction between the physical and the cyber element, and not only the physical and computational elements separately [105, 89].

As CPPS have the purpose of quickly adapting to new requirements such as new products or product variants, the automation system requires high manual engineering efforts for every new adaption [120]. After an adaption, the parameters of the system need (re-) optimization to run efficiently. Formally, we search arguments of a function f, realized by the CPPS, that optimizes (minimizes, without loss of generality) this function:

$$\arg\min_{x \in \mathcal{X}} f(x), f\colon \mathcal{X} \to \mathbb{R} \tag{1.1}$$

The set $\mathcal{X} \subseteq \mathbb{R}^d$ is the non-empty, parameter space, and each element $x \in \mathcal{X}$ represents a candidate solution. The parameters can broadly be classified into discrete, categorical, or continuous and the problem can also be of mixed parameter types. In this thesis, we develop solutions with a focus on continuous parameters.

A suitable algorithm must be selected to solve this problem efficiently. The problem of selecting an efficient algorithm for an optimization problem at hand is called the Algorithm Selection Problem (ASP) and is defined as follows [133]:

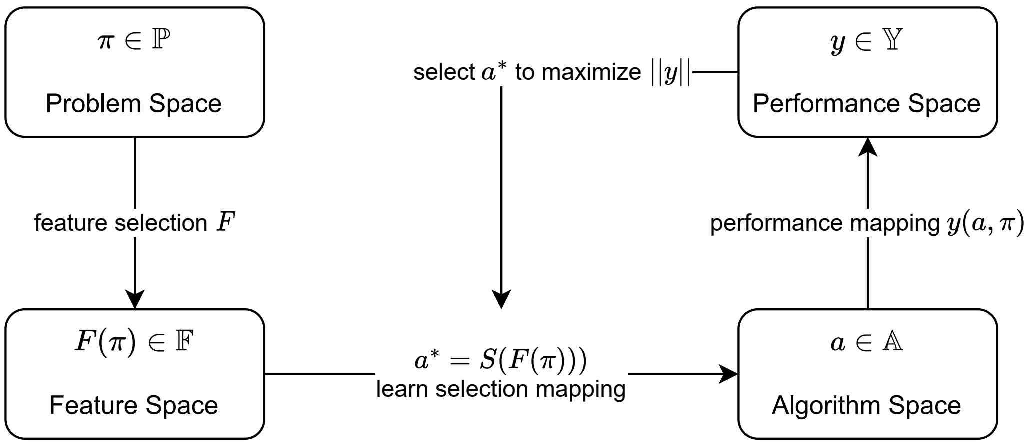

Figure 1.1: *Schema of the algorithm selection problem defined by Rice (1976). The goal is to select an algorithm that maximizes some performance metric on a given problem. The figure is based on [141] and was adapted to the notation of the thesis at hand.*

Definition 2 (Algorithm Selection Problem)**.** *For a given problem instance $\pi \in \mathbb{P}$, with features $F(\pi) \in \mathbb{F}$, find the selection $S(F(\pi))$ into the algorithm space $\mathbb{A}$, such that the selected algorithm $a \in \mathbb{A}$ maximizes the performance $y(a, \pi) \in \mathbb{Y}$.*

The schematic of the problem is shown diagrammatically in Fig. 1.1. The selection procedure starts with a problem instance π (which represents the function $f(x)$ from Eq. 1.1) and the computation of features $F(\pi)$, which are then used as arguments for the selection mapping function. This function selects an algorithm a that maximizes a performance metric on the problem π.

In general, Machine Learning (ML)-techniques seem suitable to address the ASP. Classifiers can be trained to learn the decision boundaries for efficient algorithms on the problems by modeling the selection mapping function according to a defined performance metric and algorithm portfolio. Exploratory Landscape Analysis (ELA) can be used to compute numerical features of problem instances [99, 101]. A review of the current research on ASP methods and examples of a combination of ELA and ML can be found in the literature [132, 110, 113, 80].

However, both approaches currently have some drawbacks or are even infeasible for the described online scenario, as a significant number of training samples are needed, i.e., many results of different algorithms on optimization problems with a certain similarity to the problem at hand w. r. t. its features. Additionally, recent research indicates that computation and selection of relevant ELA features can become quite challenging. It is sensitive to both the sampling strategy applied and the number of samples [131, 109]. Furthermore, not all features are invariant to simple transformations of the fitness landscape, i.e., shifting and scaling [85], which is quite possible when the CPPS is adapted.

Our approach to addressing the ASP will use a data-driven test instance generation procedure based on Gaussian process model (GPM) to evaluate algorithm performance in an online procedure, i.e., during the runtime of the CPPS, since data may not be available beforehand to perform offline algorithm selection.

We will especially consider the requirements of Small and Medium-sized Enterprises (SMEs). Concrete implementations of solutions, even if they address problems in a generalizable way, are tied to a specific platform or technology. This means that they cannot be used by all companies or may even become unusable in the future, i.e., they rely on subsequent development in terms of bug fixes and adaptation to infrastructure changes. To provide added value, the goal is to develop an architecture that addresses the problem of automatic online algorithm selection for CPPS optimization. This will ensure cost savings when implementing the targeted solution or parts of it. The saved effort increases if several use cases are implemented, e.g. optimization of several machines.

Applications and architectures for CPPS are often associated with the term big data due to the availability of increased volume, velocity, and variety of data. For a definition of big data, we cite De Mauro et al.[38, p.131]:

Definition 3 (Big Data)**.** *"Big Data is the Information asset characterised by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value."*

This raises challenges for organizations in terms of data management, processing and storage, as well as requirements for solutions such as scalability to maintain performance. However, the intended solution for the ASP will only be slightly affected by the topic of Big Data, as we do not aim to develop special algorithms dealing with Big Data, but rather focus on embedding solutions for process parameter optimization in general in a Big Data context, i.e. in a Big Data architecture and a prototypical implementation for solving optimization problems in CPPS.

1.2 Research Questions

Considering the schema of the ASP shown in Fig.1.1, there are several open questions and challenges to be addressed in this thesis. The generation of data-driven test instances during the runtime of the CPPS based on GPM needs to be validated, i.e. to ensure feasibility in terms of runtime and accuracy on the one hand. The generation of data-driven test instances during the runtime of the CPPS based on GPM must be validated, i.e., are they feasible in terms of runtime and accuracy, or are there already reliable test instances for the optimization problems implemented by CPPS? These questions refer to the problem space $\mathbb{P}$ and the feature space $\mathbb{F}$ of ASP.

Furthermore, the solution should adapt to new, previously unseen functions automatically, i.e. without the need for additional expert knowledge to select a suitable algorithm. This thesis addresses the question of how to measure the performance of algorithms and how to transfer the necessary knowledge to achieve an automated solution. These questions are related to the specification of the algorithm space $\mathbb{A}$, the performance space $\mathbb{Y}$, and the selection mapping function $S(F(\pi))$. Additionally, open questions regarding architectural aspects will be addressed. This These questions are summarized below:

(RQ-1) Are Gaussian process simulation (GPS) and model variations feasible and efficient to allow online algorithm selection, and what are the boundaries for an efficient implementation?

The application of test instance generation based on Gaussian process (GP) requires an analysis of the relationship between test instances generated by different methods and their parameterizations, as well as insight into the impact of generation method and configuration on the performance of optimization algorithms.

(RQ-2) How can a solution for the ASP be algorithmically implemented, such that operators can optimize a CPPS online with a minimum of data science knowledge and without a hand-written procedure?

The shortage of skilled workforce affects engineering disciplines, leading to more responsibilities for employees regarding monitoring and optimizing production processes and data science and computer science. The main focus regarding this question in this thesis is the development of an algorithmically exploitable knowledge base for both optimization problems and optimization algorithms. Important questions are what kind of knowledge and information are needed to implement a relationship between optimization problems and algorithms. This should finally lead to an application without the need for monolithic solutions to implement an optimization algorithm to optimize a CPPS.

(RQ-3) How can the data-driven online algorithm selection solution be addressed by a system architecture that provide benefit for many companies and what are the requirements for an implementation?

Since not every company, especially SMEs, has common requirements regarding computing infrastructure or even operating system platforms, the definition of an architecture can greatly reduce the implementation effort when specific requirements have to be addressed. The level of abstraction at which an architecture can be defined ranges from e.g. reference architectures, which provide templates and processes for defining software, to software architectures, which represent a tailor-made solution for a specific company and a specific use case. Our goal is to develop a system architecture that specifies the structure and behavior of the necessary components to address CPPS use cases with a focus on process parameter optimization. Such an architecture will provide maximum benefit if it can be used by many companies for a large number of use cases. We will discuss and address the balancing act between the level of abstraction and the effort required for implementation.

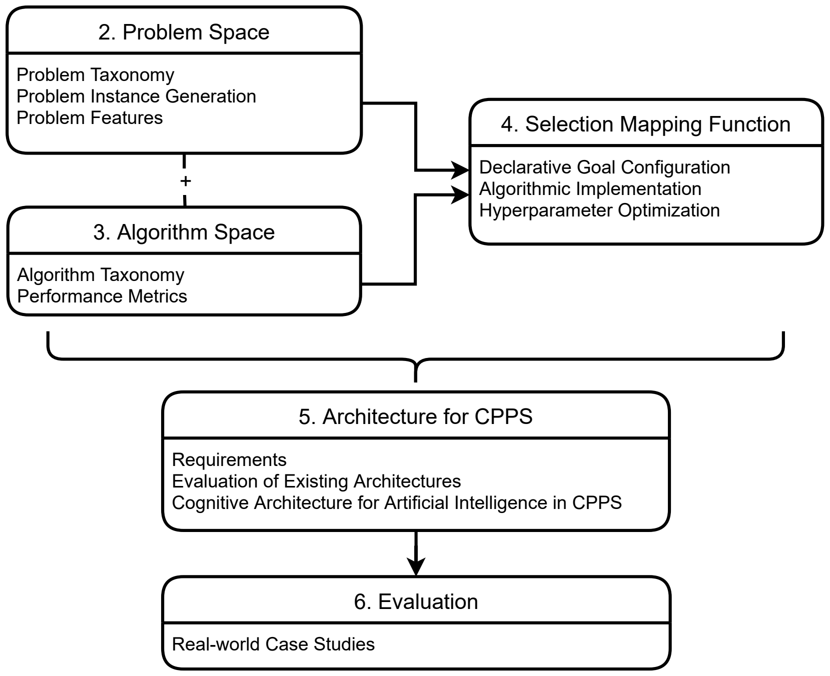
2. Problem Space
Problem Taxonomy
Problem Instance Generation
Problem Features
+
3. Algorithm Space
Algorithm Taxonomy
Performance Metrics
4. Selection Mapping Function
Declarative Goal Configuration
Algorithmic Implementation
Hyperparameter Optimization
5. Architecture for CPPS
Requirements
Evaluation of Existing Architectures
Cognitive Architecture for Artificial Intelligence in CPPS
6. Evaluation
Real-world Case Studies

Figure 1.2: *Graphical summary of the contents of this thesis. The numbers represent the chapters.*

1.3 Outline

An overview of the contents of this thesis is summarized in Fig. 1.2. The first three chapters following this introduction are dedicated to the different spaces of the ASP and build the fundament for the developed architecture.

Chapter 2 provides insight into the problem space specification. Section 2.1 presents the taxonomy for problem classes in CPPS used in this thesis. The introduction to GPM and three different strategies for generating test instances based on GPM are given in Section 2.2. In Section 2.3, the generated test instances are evaluated with respect to their features and the performance of the algorithms on different problems of the injection molding simulation use case. Chapter 3 covers the algorithm taxonomy used in this thesis. Section 3.1 provides an overview of different algorithm families and in Section 3.2 we discuss several relevant performance metrics. The selection mapping approach is described in Chapter 4. Section 4.1 deals with the structure of the knowledge base that is needed to define the problems and accordingly select instances from the available algorithm portfolio. The algorithmic implementation of the selection mapping function is given in Section 4.2, and the applied method for hyperparameter optimization of the algorithms is described in Section 4.3.

Chapter 5 introduces the developed architecture, which combines and implements the solutions and approaches of the previous chapters. The requirements for our architecture, which address CPPS use cases arising from selected use cases and general considerations are provided in Section 5.1. Several selected reference architectures from the fields of automation and cognitive science are discussed and evaluated according to the previously collected requirements in Section 5.2. Section 5.3 presents the newly developed architecture for CPPS.

The developed solutions are finally evaluated and discussed in Chapter 6. Two real-world case studies are presented. In Section 6.1, we present the results of Case Study 1, the Versatile Production System. The results of Case Study 2, Injection Molding Optimization, are presented in Section 6.2.

This thesis concludes with a final evaluation in Chapter 7.

1.4 Publications

The following publications are of relevance to this thesis, as they comprise important results and findings to the research topic. Parts of these papers were extended, restructured, and rewritten where needed to give more detailed information or extend the experiments where necessary. Some parts were used verbatim in this thesis. The following listing is presented based on its publication date, and the main contribution of the author of this thesis is given for each item.

Data-driven Problem Classification and Algorithm Selection for Injection Molding Optimization, A. Fischbach, F. Rehbach, D. Anders, Materials, Methods & Technologies, Volume 16, **2022** [52].

- The author of this thesis developed and implemented a strategy to evaluate several GPM-based test instance generation methods on an injection molding simulation use case for algorithm selection.

Benchmark-Driven Algorithm Configuration Applied to Parallel Model-Based Optimization, F. Rehbach, M. Zaefferer, A. Fischbach, G. Rudolph, and T. Bartz-Beielstein, IEEE Transactions on Evolutionary Computation, **2022** [130].

- The author of this thesis contributes to the benchmarking concepts implemented for algorithm configuration based on simulations, problem features, and experimental design considerations.

Cognitive Capabilities for the CAAI in Cyber-Physical Production Systems, J. Strohschein, A. Fischbach, A. Bunte, H. Faeskorn-Woyke, N. Moriz, and T. Bartz-Beielstein, The International Journal of Advanced Manufacturing Technology, **2021** [146].

- The author of this thesis contributed the algorithmic concepts and solutions for the online algorithm selection problem, corresponding benchmark experiments, and the python implementation of the algorithms and the algorithm selection procedure.

Benchmarking in Optimization: Best Practice and Open Issues, T. Bartz-Beielstein, C. Doerr, D. van den Berg, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, W. La Cava, M. Lopez-Ibanez, K. M. Malan, J. H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise, arxiv e-Print, **2020** [8].

- The author of this thesis contributed to Section *Algorithms* in cooperation with C. Doerr, and contributed fundamental ideas to Section *Experimental Design*.

CAAI-a cognitive architecture to introduce artificial intelligence in cyber-physical production systems, A. Fischbach, J. Strohschein, A. Bunte, J. Stork, H. Faeskorn-Woyke, N. Moriz, and T. Bartz-Beielstein, The International Journal of Advanced Manufacturing Technology, **2020** [53].

- The author of this thesis contributed significant parts to the development of the architecture and the prototypical implementation and evaluation on a real-world use case based on experiments comparing two Surrogate Model-Based Optimization (SMBO) variants w.r.t. performance and resource consumption.

Improving the reliability of test functions generators, A. Fischbach and T. Bartz-Beielstein, Applied Soft Computing, **2020** [51].

- The author of this thesis contributed to the provided problem taxonomy, the experimental setup, and the evaluation of the referenced random and mixed ANOVA models for selected problem and algorithm classes.

Open Issues in Surrogate-Assisted Optimization, J. Stork, M. Friese, M. Zaefferer, T. Bartz-Beielstein, A. Fischbach, B. Breiderhoff, B. Naujoks, and T. Tušar, in High-Performance Simulation-Based Optimization, Springer, **2020** [145].

- To this book chapter, the author of this thesis contributed to the section *Benchmarking* in cooperation and discussion with M. Zaefferer.

Evaluation of Cognitive Architectures for Cyber-Physical Production Systems, A. Bunte, A. Fischbach, J. Strohschein, T. Bartz-Beielstein, H. Faeskorn-Woyke, and O. Niggemann, in 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), **2019** [23].

- The first three authors share the first authorship with an approximately equal share of the work. The author of this thesis contributes the evaluation of the Industrial Internet Reference Architecture (IIRA), the evaluation of the Adaptive Control of Thought-Rational (ACT-R) architecture, the requirements for a new architecture based on the evaluation of an energy efficiency optimization use case, and general conceptual requirements.

Simulation Based Test Functions for Optimization Algorithms, M. Zaefferer, A. Fischbach, B. Naujoks, and T. Bartz-Beielstein, in Proceedings of the Genetic and Evolutionary Computation Conference, **2017** [155].

- The author of this thesis contributed to the experimental setup and general ideas for using and applying gaussian process simulations. The simulation-based test function generation for discrete optimization problems was developed and implemented by M. Zaefferer.

From Real World Data to Test Functions, A. Fischbach, M. Zaefferer, J. Stork, M. Friese, and T. Bartz-Beielstein, in 26th Workshop Computational Intelligence Proceedings, **2016** [54].

- The author of this thesis developed and implemented the evaluation of a parameter variation strategy on GPM parameters to retrieve and compare real-world data-based test instances for benchmarking using an ensemble of similarity metrics.

Chapter 2

Problem Classes and Problem Features

This chapter introduces a problem taxonomy for this thesis and discusses different points of view on defining problem classes. In theory, the No-free Lunch Theorem (NFL) [150] provides a significant limitation for selecting and tuning global optimization algorithms for a problem at hand. First, it is not realistic to have one single algorithm that performs superior on all problem classes. If one algorithm performs superior on one particular set of objective functions, it must perform correspondingly poorly on other objective functions [149]. Additionally, improving an algorithm, i.e., by tuning its parameters, for one class of problems likely makes it perform more poorly for other problems. As a practical consequence, we focus on well-defined problem classes.

Accordingly, Haftka recommends that optimization algorithms "should be targeted towards a particular application or set of applications rather than tested against a fixed set of problems." [60, p. 1]. Recommendations according to the consequences of the NFL for optimization, as described in [8], contain:

a) bound claims of algorithm performances to the tested instances,

b) matching suitable algorithms to problem classes is a good thing,

c) be cautious about generalizing performance to other problem instances, and

d) be very cautious about generalizing performance to other classes or domains.

This chapter provides the terminology to specify the problem class $\mathbb{P}$ for real-world problems in the context of Cyber-physical Production System (CPPS). The goal is not to find the best or formally correct definition of a problem class, as there are many possible ways and perspectives. Instead, the goal is to find a description that is beneficial to adequately address the Algorithm Selection Problem (ASP) for CPPS. Consequently, this chapter describes methods to classify problem instances according to a given real-world process as narrowly as possible to retrieve relevant instances. This taxonomy will be used as a frame for the boundaries of generalizability of algorithms' performances.

Bartz-Beielstein et al. emphasize the significance of a problem taxonomy in the context of benchmarking optimization algorithms [8]. They provide a slightly different terminology with one additional level of abstraction compared to the taxonomy used in the thesis at hand (see Fig. 2.1).

Regarding easy or cheap to evaluate objective functions, many testbeds are available. Arguably, one of the most widely used is the Black-Box Optimization Benchmarking suite (BBOB) [63]. The single-objective subset of the BBOB consists of 24 test problems divided into five classes. Users of the BBOB can scale each test problem in dimensionality and randomize problems w.r.t. translation and rotation. Furthermore, the landscape and optima of each of the problems are known. A detailed overview of the problems contained in this benchmarking suite can be found in [64]. Besides their beneficial features, testbeds like BBOB might also have some drawbacks, depending on the use case. Even though benchmark sets are developed with a larger coverage of the single-objective, bound-constrained problem space [88], their problem classes may not be relevant to the problem at hand. Muñoz and Smith-Miles generate a more diverse testbed using genetic programming [111]. Another approach by Dietrich and Mersmann, which addresses the lack of diversity in existing testbeds, uses affine recombinations of pairs of BBOB functions to generate additional benchmark functions [43]. Although both approaches are specifically helpful for the benchmarking discipline and the research of optimization algorithms in general, regarding the ASP for a specific problem class, they do not guarantee the availability or the ability to generate relevant test instances. Performing benchmark experiments for algorithm selection might then be misleading. Furthermore, as conditions in CPPS under dynamic changes affect the optimization problems landscape, it is unclear if the testbed covers the necessary degree of variety and thus leads to the selection of a non-optimal algorithm. Ultimately, generating relevant test instances based upon recent data of the CPPS can address this issue.

2.1 Problem Classes

The taxonomy for optimization problems used in this thesis is depicted in Fig. 2.1:

Problem Instance A problem instance is a concrete function to optimize, i.e., the gaol is to find the arguments that yields the minimum function value, see Eq. 2.1. In the context of injection molding, it is given by a specific mold (the geometry of the parts to produce), raw material, and environmental properties like humidity, room temperature, or the machine's condition.

Problem Class A problem class is the set of problem instances generated by a concrete machine or production process.

Problem Family A problem family represents a type of a production process, such as injection molding.

This thesis aims to find a suitable, robust, and efficient algorithm for an a priori unknown problem instance of a given single-objective, continuous problem class. A problem instance is defined as a function f which is to be minimized:

$$\arg\min_{x \in \mathcal{X}} f(x), f\colon \mathcal{X} \to \mathbb{R} \tag{2.1}$$

The set $\mathcal{X} \subseteq \mathbb{R}^d$ is the non-empty, continuous parameter space, and each element $x \in \mathcal{X}$ represents a candidate solution. The dimensionality d of the parameter space is determined by the CPPS. This parameter space is likely not unlimited in terms of its boundaries. They are constrained to a given range, especially for real-world applications, where the parameters also describe physical quantities. To operate a CPPS safely, bound constraints are defined and must be respected by algorithms during parameter adaption.

Consequently, we do not expect to transfer locations of optima between different problem instances. However, algorithms that efficiently optimize one instance will likely be similarly efficient on a different instance of the same problem class.

If we look at a certain CPPS and consider that the raw material is changed, or at least some of its properties (e.g., moisture), the problem instance will also change. Changes in the objective space might only be slight, and several features of the problem will be preserved. The search space is considered mostly unchanged. The bound constraints may slightly change, e.g., if a new raw material does not allow certain temperatures. However, the fundamental physical behavior and rules are likely similar. An example of different instances of artificially generated problem classes generated with the Gaussian Landscape Generator (GLG) [57] retrieved with the same configuration for each class is shown in Fig. 2.2.

With only setting a few parameters, the GLG can be used to set up problem instances for continuous, bound-constrained optimization problems. These instances are variable w.r.t. the number of dimensions, the number of Gaussian components to be placed randomly, which controls the modality, and the Region of Interest (ROI). The

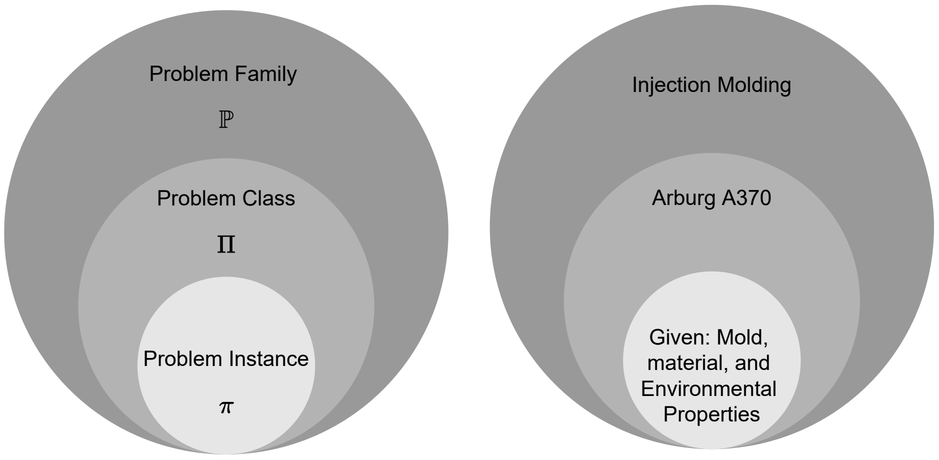

Figure 2.1: *Schema of the Problem Taxonomy used in this thesis. The general hierarchy is given on the left, and a real-world example is shown on the right. The Problem Family represents the production process in general. A particular problem class is given by an existing machine that implements a certain type of this process. A problem instance as the smallest problem entity represents an optimization problem given by an explicit machine instance, a given type of raw material with its current condition (e.g., the material moisture), and the current environmental properties (e.g., the temperature and humidity in the room). The Figure is based on [9].*

user can specify the ratio between the value of the global optimum and the local optima. This enables the creation of many test instances with a similar structure, representing a problem class.

An appropriate algorithm for a problem class should perform well on most instances to address the expected variance. Consequently, different instances of one specific problem class should reflect the natural variance of the process (pure repetition error), different material properties, and environmental properties.

Table 2.1 summarize the expected relative difference in search space and objective space between different problem instances, classes, and families.

The primary goal of this thesis is to find an algorithm suitable to robustly optimize the problem class, not only a single instance. Several findings from the field of benchmarking should be taken into consideration [8]: The experimenter must choose both the test problems and the algorithms to benchmark with care. Additionally, statements regarding the generalizability of the performances of the algorithms will be limited to the problem class at hand. The degree of similarity w.r.t. the problem characteristic, i.e., the features of the problem, will be described in Section 2.3.

It is common benchmarking practice, e.g., when new algorithms are introduced and applied to a problem suite, to select problem instances from different problem classes and maybe even different problem families. Contrarily, for the task of selecting a proper algorithm for a specific CPPS, we focus on problem instances from one problem class, namely the class representing the CPPS at hand. This impacts the requirements for the desired set of problem instances. Instead of being space-filling or

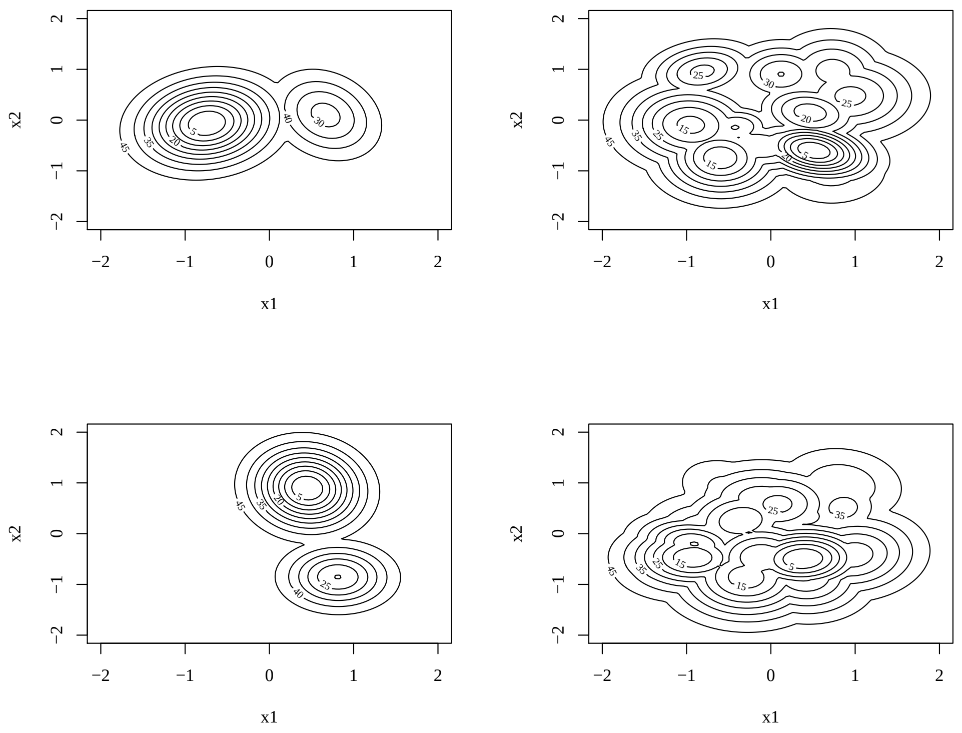

Figure 2.2: *Contour plot of four different instances retrieved from the Gaussian landscape generator. The Parameter space (x1, x2) is shown on the x- and y-axis, and the function value is indicated as a height value on several height levels. The value of the global optimum is* 0*. A column-wise significant difference between instances w.r.t. at least some landscape features can be recognized.* Left column: *Problem class* Π_1*. Two random instances with the number of Gaussians set to 2 and the ratio between global and local optima set to 0.8.* Right column: *Problem class* Π_2*. Two random instances with the number of Gaussians set to 25 and the ratio set to 0.8. The Figure is taken from [51, p. 5].*

diverse, they should reflect features of the addressed problem class. Arguably, it is hard to select relevant test problems when the class is unknown or new. So, for the goal of algorithm comparison, labeled as (G1.2) in Bartz-Beielstein et al. [8], we suggest the data-driven generation of test instances.

Another way, which is different from the focus of this thesis, could be to evaluate the current machine, compute features of the problem, and select available test instances with similar features. Available literature shows that existing test suits do not cover all regions of the feature space very well; see, e.g., [111, 94, 8]. Consequently, there is no guarantee that suitable instances are available. Therefore we focus on the generation of test instances with as few evaluations of the problem and computational effort as possible w.r.t. the desired quality of the instances. This is important, as the process, i.e., the CPPS, may be adjusted over time according to market demands and

Table 2.1: *Expected degree of relative changes between different problem families, different classes from the same family, and different instances from the same problem class w.r.t. search space and objective space.*

	Search Space	**Objective Space**
Problem Family	typically large differences	possibly large difference of features
Problem Class	typically small to medium differences	possibly small to medium differences of features
Problem Instance	typically similar or equal	important features are possibly similar or equal

new developments. With a higher degree of automation, changes in the process can occur more quickly. In the example of injection molding, molds can be produced ad hoc by, e.g., an additive manufacturing process. An online solution to optimize this process must therefore be able to adjust to the expected degree of change and accordingly select an appropriate algorithm frequently.

2.2 Problem Instance Generation

This section is partly based on the following publications [54, 155, 130]. The notation and structure were adapted to this thesis, and some text parts were taken verbatim.

As motivated previously, an online solution for the algorithm selection problem can significantly benefit from a data-driven model-based generation of problem instances for algorithm hyperparameter optimization and algorithm selection. We aim to generate problem instances that preserve characteristics of the real-world problem at hand.

In this thesis, we consider Gaussian process models (GPMs) to model the process, as they fit very well in the context of CPPS. They can be very accurate and do not need expert knowledge about the process to model, i.e., the model structure must not be defined a priori. Furthermore, they provide a measure for uncertainty, which can be used to balance exploitation and exploration on the one hand and variations on the other. However, GPMs are not parameter-free, as a kernel must be chosen or can automatically be selected by, e.g., applying cross-validation during the training process. Additional parameters can be fitted employing numerical optimization during the training process, e.g., the regularization constant λ, also called *nugget*, for noisy data.

2.2.1 Gaussian Process Models

GPMs constitute a class of probabilistic statistical models, which can be used, e.g., for regression and interpolation. They are based on Gaussian processs (GPs), which describe the uncertainty about an underlying process. Consider a sampling plan $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1\dots n}$ in a k-dimensional continuous search space, leading to n observations $\mathbf{y} = \{y^{(i)}\}_{i=1\dots n}$, a GPM tries to predict values at an unknown location in the search space by interpreting the observations $\mathbf{y}$ as realizations of a stochastic process. This stochastic process is defined by the set of random vectors $\mathbf{Y} = \{Y(\mathbf{x}^{(i)})\}_{i=1\dots n}$. The correlation of the random variables $Y(\cdot)$ is modeled as follows [56]:

$$\text{cor}\left[Y(\mathbf{x}^{(i)}), Y(\mathbf{x}^{(l)})\right] = \exp\left(-\sum_{j=1}^{k} \theta_j |x_j^{(i)} - x_j^{(l)}|^{p_j}\right) \tag{2.2}$$

The matrix that collects correlations of all pairs $\{(i,l)\}$ is called the correlation matrix $\mathbf{\Psi}$. It is used in the predictor

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \boldsymbol{\psi}^T \mathbf{\Psi}^{-1}(\mathbf{y} - \mathbf{1}\hat{\mu}) \tag{2.3}$$

where $\hat{y}(\mathbf{x})$ is the predicted function value of a new sample $\mathbf{x}$, $\hat{\mu}$ is the Maximum Likelihood Estimate (MLE) of the mean and $\boldsymbol{\psi}$ is the vector of correlations between training samples $\mathbf{X}$ and the new sample $\mathbf{x}$. The width parameter $\boldsymbol{\theta} = \left(\theta_1, \dots, \theta_j, \dots, \theta_k\right)^T$ determines how far the influence of each sample point $\mathbf{x}$ spreads. In detail, the larger the width parameter, the faster the potential changes in the predicted value. The smaller the width parameter is, the slower the potential changes in the prediction are. Since there is one θ_i for each dimension, this parameter accounts for the activity in each dimension. The parameter p_j is usually fixed at $p_j = 2$, and defines the shape of the correlation function: At $p_j = 2$, the correlation function is more smooth, whereas $p_j = 1$ is less smooth. If the process is considered noisy, the regularization parameter λ is added to the diagonal of the correlation matrix $\mathbf{\Psi}$. This enables the model to avoid overfitting, which can occur when all training data are reproduced exactly (interpolation), i.e., the model would not be able to discriminate between the underlying response and the noise. Classically using λ to deal with noisy data, it can also smoothen more rugged fitness landscapes. All model parameters, e.g., $\boldsymbol{\theta}, \lambda$ and $\boldsymbol{p}$, are determined by MLE, which requires numerical optimization.

2.2.2 Generation Process

To start filling the gap of missing real-world relevant problem instances for a given process, a data-driven approach for the generation of instances based on real-world data was published in [54]. It consists of the following subsequential steps.

1. First, a data set of a real-world problem is collected, consisting of a design matrix $\mathbf{X}$ and a vector of corresponding outcome values of the underlying process$\mathbf{Y}$.

2. Then a model with regression and interpolation capabilities is built, further denoted as *level 0 model*. In this work, a GPM is used as the *level 0 model*.

3. Next, a parameter (scalar or vector) is defined to create variations of the previously fitted *level 0 model*. This parameter is used to perturb the generated model, and generate n test instances, further denoted as *level 1 models*, which are in turn variations of the *level 0 model*. Finally, a randomly selected subset of the desired number of $m < n$ instances is chosen for benchmarking experiments.

In the following we discuss three different strategies to generate GPM variations and their parameterization. The first strategy *Parameter Variation* (Strategy 1) uses explicit variations of the model parameters θ_i and λ, the second strategy *One Stage Simulation* (Strategy 2) uses Gaussian process simulation (GPS), and the third strategy *Two Stage Simulation* (Strategy 3) uses the aggregation of two GPS models, one accounts for the global structure, and a second model adds finer local structure.

2.2.3 Strategy 1: Parameter Variation

The Parameter Variation (PV) strategy aims at directly varying model parameters. The main parameters controlling the behavior of the GPM are arguable λ and $\boldsymbol{\theta}$. The primary goal of the test instance generator is the deployment upon real-world data, which is usually noisy. Furthermore, repetition data may not be available, e.g., due to its cost. Consequently, the variation of the parameter λ is a natural first choice. In addition, the variation of the width parameter θ appears important to change the model to the desired degree while maintaining the general characteristic of the fitness landscape under certain circumstances. The bounds of the parameters' variations must be defined carefully and adapted to the problem. Otherwise, the model can show signs of degeneration.

The test function generator will compute lower and upper bounds for the variation of the parameter according to the fitted *level 0 model*. We define a variation vector $\vec{\alpha}$ as follows: The first component of α represents the offset of the λ value, and the remaining components represent the offsets for corresponding $\boldsymbol{\theta}$ values. They will further be denoted as λ_S and θ_{Si} respectively. They will be added to their corresponding values of the *level 0 model* to retrieve the altered *level 1 model*. Afterward, the correlation Matrix $\boldsymbol{\Psi}$ is recalculated for predictions, realizing the desired test instance.

This approach addresses two relevant problems: First, if a model shows insensitivity to some parameter, which is varied, derived test instances might be very similar. Second, if one parameter is highly sensitive and dominating, a random change can lead to irrelevant test instances without resembling the underlying problem.

The variation parameter α is defined as: $\vec{\alpha} = (\lambda_S, \theta_{Si})$ for $i = 1...n$, where n is the dimensionality of the problem, λ_S the λ value for the simulation, and the θ_{Si} the values for dimension i for the simulation.

2.2.4 Strategy 2: One-Stage Simulation

Contrary to a GPM prediction, where the goal is to estimate a function value at a new sample x as close as possible to the actual value, a simulation tries to produce values whose moments are as close to the moments of the actual data as possible [77]. The simulation approach based on GPM creates realizations of a GP with the same mean and covariances as the *level 0 model*.

The approach described here is based on the square root of the covariance matrix [34] due to computational reasons, although different methods exist. Firstly, a set $\boldsymbol{X_s}$ of m samples is selected. The process will be simulated at these samples. Secondly, the correlation matrix $\boldsymbol{K}_s$ of the set $\boldsymbol{X_s}$ is computed. It is decomposed as $\sigma^2\boldsymbol{K}_s = \boldsymbol{C}_s = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, where the process variance σ^2 is determined by MLE and $\boldsymbol{C}_s$ is the covariance matrix with the eigenvector matrix $\boldsymbol{U}$ and diagonal eigenvalue matrix $\boldsymbol{\Lambda} = \text{diag}(\boldsymbol{\lambda})$. The square root of $\boldsymbol{C}_s$ is $\boldsymbol{C}_s^{1/2} = \boldsymbol{U}\text{diag}(\lambda_1^{1/2}, ..., \lambda_m^{1/2})\boldsymbol{U}^T$ and the simulation is given by

$$\hat{\mathbf{y}}_{nc} = \mathbf{1}\hat{\mu} + \boldsymbol{C}_s^{1/2}\boldsymbol{\epsilon} \tag{2.4}$$

Here, $\boldsymbol{\epsilon}$ is a vector of m independent, normally distributed random numbers with zero mean and unit variance.

The conditional simulation aims to reproduce the moments of the training data and simultaneously exactly reproduce the training data. The conditional simulation may result in more realistic shapes than the predictor. For example, higher frequency behavior may not be visible in the predictor but may be visible in a (conditional) simulation [34]. Compared to the previously described *parameter variation* variant, this can be an advantage.

The simulations based on decomposition rely on selecting a set of samples distinct from the observed training samples, where the process is then simulated. The simulation samples can (and should) largely exceed the training samples, so this can quickly become computationally infeasible, depending on the requirements regarding the response time. If the search space gets larger, the need for interpolation between the simulation samples rises, again leading to undesired smoothing of the test instance. Consequently, we recommend the decomposition-based approach, especially for smaller discrete search spaces.

We follow the spectral method for continuous search spaces, see [34]. It yields a function that is evaluable at any desired location without interpolation. The main idea is to superimpose cosine functions to retrieve the simulation:

$$f_S(x) = \hat{\sigma}\sqrt{\frac{2}{N}}\sum_{v=1}^{N} cos(\omega_v \cdot \mathbf{x} + \phi_v) \tag{2.5}$$

where ϕ_v is an i.i.d. uniform random sampled value from the interval $[-\pi, \pi]$, and $\omega_v \in \mathbf{R}^n$ are i.i.d. random samples from a distribution with a density equal to the one from the spectral density function of the GPM's kernel. Using the kernel from Eq. 2.2, this distribution is the normal distribution with zero mean and variance $2\sigma_i$, for dimension i.

Conditional simulations, i.e., simulation functions that are conditioned to the training data, can be performed with $f_{SC}(\mathbf{x}) = f_S(\mathbf{x}) + \hat{y}^*(\mathbf{x})$.
$\hat{y}^*(\mathbf{x}) = \mu + \mathbf{k}^T\mathbf{K}^{-1}(\mathbf{y}_{SC} - 1\hat{\mu})$ is the predictor defined in Eq. 2.3, with $\mathbf{y}$ replaced by values from the unconditional simulation $\mathbf{y_{SC}}$, i.e., $y_{SCj} = f_S(\mathbf{x_j})$, and $\hat{\mu} = 0$.

The one stage simulation can be performed *conditional* or *unconditional*, and the parameter $Ncos$, specifying the number of superimposed cosin functions for the simulations, have to be set. In this context, *unconditional* means that the simulation reproduces only the covariance structure, but not the observed values y of the training data set.

2.2.5 Strategy 3: Two-Stage Simulation

This section is based on the current implementation and documentation of the software package Continuous Optimization Benchmarks By Simulation (COBBS) for the R platform by Zaefferer [154]. It provides tools to create simulation-based benchmarks for continuous optimization and especially implements the GPS. The concepts were first published in [156], and the extension to the two-stage (2-stage) model was published in [130].

The basic idea of the 2-stage model is to target both global and local structures of the optimization problem at hand. Using a stationary GPM with a constant global trend may limit the model quality and, likely, decrease the complexity of the retrieved test instances. It will either preserve the problem's global or local structure, as it will fit only one activity parameter θ per dimension.

The approach by Rehbach et al. [130] starts with fitting a GPM as a *first-stage model*, with enabling the regularization, i.e., the parameter λ will be fitted using MLE. This enables the model to regress and smoothen the data [56]. While this preserves the global structure, it could smoothen the local structures of the problem. To target this effect, a second GPM model is fitted to the residuals of the *first-stage model*. This *second-stage model* will function as an interpolation model, i.e., without fitting the regularization constant, aiming at preserving the local structures of the process. Simulations of the combined *two-stage model* are retrieved by adding the predictions of both simulation models. This leads to a model with a non-stationary trend, which can be varied for the simulation and works as the *level 0 model*. Similar to the 1-stage simulation, the 2-stage simulation can be performed *conditional* or *unconditional*, and the parameter $Ncos$ have to be set.

Examples

To get a visual expression of what kind of test instances can be generated with the described variation strategies, Fig. 2.3 depicts several one-dimensional test instances generated based on the same *level 0 model*. The ground truth is described by the function $f(x) = sin(25 * x) + sin(54 * x - 0.2) + x$. The *level 0 model* is most accurate in regions where the search space is more exhaustively sampled, i.e., where

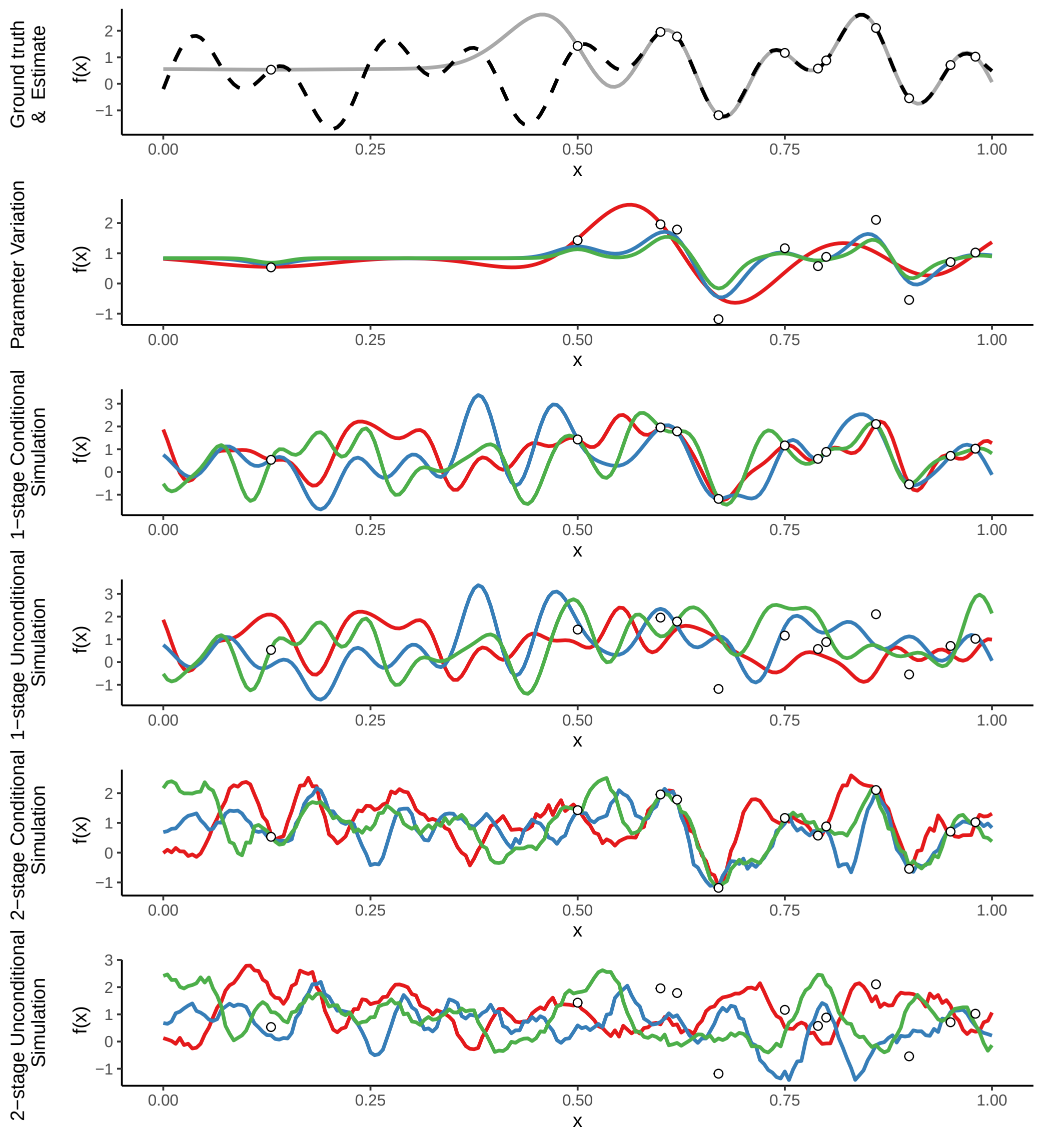

Figure 2.3: *The top rows show the ground truth function (dashed line), the training data (circles), and the GPR model estimation (solid gray line). Row-wise followed by three instances of each variation strategy, namely the parameter variation (slight changes to λ and θ), 1-stage simulation (conditional and unconditional), and 2-stage simulation (conditional and unconditional). Each generated test instance is based on the same level 0 model. The Figure is based on [156].*

$0.5 \leq x \leq 1.0$. The GPM tends to smoothen the function recognizably in sparsely sampled regions. The PV instances largely reproduce the characteristic of the *level 0 model*, while the 1-stage and 2-stage simulations extend the characteristics of the well-known regions to the sparser sampled regions. The conditional simulations not only reproduce the characteristics of the function, but also respect the training data. This is important for test instances used to estimate the outcome of optimizers in terms of function values. Contrarily, the unconditional simulations appear to be rather erratic. Additionally, the 2-stage model adds some ruggedness to the function. At this moment, it needs to be clarified which variation strategy is the best choice for benchmarking experiments with the goal of algorithm selection or parameter tuning (or both). The PV approach decreases the objectives' complexity, while the GPS, especially the 2-stage simulation, might contrarily increase it. This will be further examined in the next section.

2.3 Problem Instance Validation

Parts of this section are based on the publication [52]. The paper was largely structurally adapted and extended by the parameter variation instance generation method. Parts of the paper were taken verbatim and included in this section.

This section analyzes the objective space of the generated problem instances. There are several ways to describe optimization problems. Numerical feature descriptions can lead to a rather objective classification. One way to extract features of the problem instances is provided by Mersmann et al. by extracting properties from problems with few evaluations [100]. They introduced Exploratory Landscape Analysis (ELA) as a method to efficiently match optimization algorithms to problems. ELA characterizes optimization problems by a larger number of numerical feature values that are grouped into categories [99]. These so-called *low-level features* are determined by numerical computations based on the sampling of the decision space of the problems. They describe properties like convexity, curvature, local search, and y-distribution, with a total of 50 numerical sub-features to characterize the problem structure. The advantage of this *low-level features* is that they can be determined automatically, although they are related to some *high-level features*, whose creation requires expert knowledge. This *high-level features* describe, e.g., the level of multimodality, the global structure, the separability, and homogeneity in a rather subjective manner and result in a classification schema.

To provide an insight into the generated problem instances, created with the three suggested variants (PV, 1-stage, 2-stage), we employ both an analysis of the landscape feature space and a performance-based analysis. The focus is on the following two important questions, formulated as subquestion to RQ-1:

(RQ-1.1) Do the different objectives, different test instance generation procedures, or different parameter settings of these procedures lead to discernible differences in instance space, i.e., do they produce different landscape feature vectors?

(RQ-1.2) Do changes in these parameter settings favor or disfavor any algorithm, i.e., does the parameterization of the test instance generation have a significant impact on the performance of the algorithms?

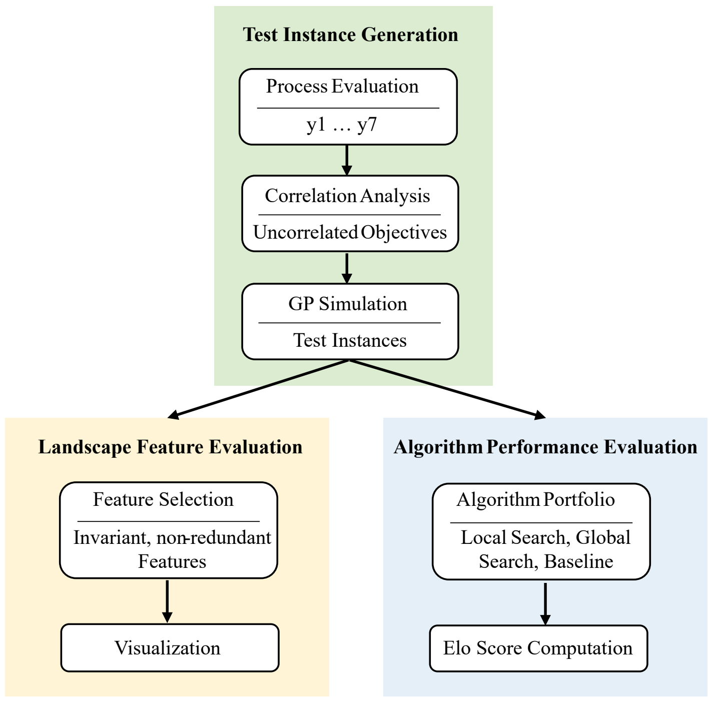

Figure 2.4: *The test instance generation for the use case of the injection molding optimization consists of three steps, namely the process evaluation, delivering the objective values $y_1 ... y_7$ for some experimental design, a correlation analysis, leading to largely uncorrelated objectives, and the GPS and GP parameter variation, leading to a set of test instances according to the process. The evaluation of the test instances consists of two procedures, the landscape feature evaluation, and the algorithm performance evaluation. The former covers the selection of invariant and non-redundant features and the 2D visualization of the test instances according to dimensionality reduction. And the latter comprises the definition of an algorithm portfolio, i.e., some local search heuristics, global search heuristics, and a baseline, and the comparison of the performances of the algorithms using Elo scores on selected test instances. This Figure is taken from [52, p. 6].*

Smith-Miles et al. introduced the Instance Space Analysis (ISA) to tackle the task of an objective algorithm performance assessment for optimization on a broader instance space instead of averaging performance across a set of chosen instances to avoid bias [140, 112]. While the feature space provides a rather algorithm-independent approach to describing problems, the performance of algorithms matters for this thesis's primary goal, i.e., selecting the most promising algorithm for the problem class at hand. So consequently, we will compare the performance of algorithms across several problems. We will also employ features of the *level 0 model* as a baseline for the ground truth. The applied validation procedure is depicted in Fig. 2.4.

The procedure of the feature selection, feature computation, feature-space visualization based on dimensionality reduction, and performance evaluation to analyze the test instance generation methods is demonstrated in the example of the Injection Molding (IM) optimization using simulations. The IM simulation is described in the next Section 2.3.1, followed by the process and the experimental setup for the analysis of the generated problem instances in Section 2.3.2. The results are summarized in Section 2.3.3

2.3.1 Injection Molding Optimization Problem

The IM process typically consists of the following steps.

1. Filling granulate in a barrel via a hopper,
2. melting the filled polymer using heaters,
3. mixing the polymer using a screw,
4. injecting the molten polymer into a mold cavity,
5. cooling the molten polymer inside the mold cavity, and
6. ejecting the solidified part(s).

The processing of these steps is driven by a set of parameter settings that should be optimized with respect to several criteria, e.g., the quality of the produced parts (in terms of dimensional accuracy, surface gloss, mechanical strength, etc.), energy consumption, and the cycle time, which also determines the production cost and throughput.

We employ Cadmould 3D-F[1] to simulate the injection molding process. Cadmould 3D-F is a commercial software that simulates the filling, cooling, and estimates the shrinkage and warpage. We use the geometry depicted in Fig. 2.5. The molted polymer is injected into the mold cavity through the sprue. The cooling channels are used to steer the cooling of the molten polymer, which in turn lead to a shrinkage of the mass. This is countered by the holding pressure and the holding pressure time, such that the mass can mostly keep its form inside the cavity. It becomes clear, that several control parameters determine the quality of an injection molding process and its produced parts.

Cadmould 3D-F employs a so-called 2,5-D approach for the filling simulation, exploiting some simplifications on the assumptions for the numerical model. This leads to a less accurate simulation but significantly lower computation time. For further details and a comparison of the Cadmould 3D-F filling simulation with a classical 3-D Computational Fluid Dynamics (CFD) simulation, we refer to [139] and Anders et al. [2]. Besides the filling, Cadmould 3D-F also simulates the cooling process to retrieve objective values regarding the quality, i.e., shrinkage, warpage, and deformation. Additionally, the required cooling time is measured, determining the cycle time.

[1]For product details, see `https://www.simcon.com/cadmould` (accessed: 2023-03-20).

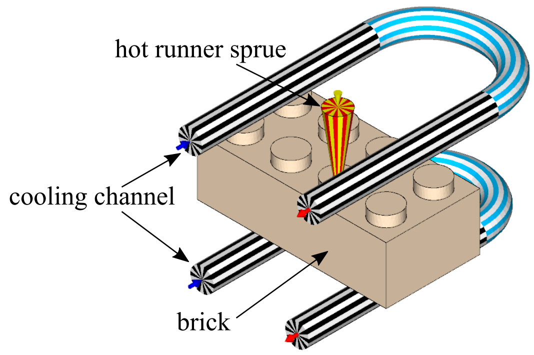

Figure 2.5: *The figure shows a 3-D representation of the injection molding process to simulate in Cadmould 3D-F. It consists of the mold cavity (the brick), the cooling channels, and the sprue.*

Regarding the optimization problem, we focus on the control parameters melt temperature (x_1), holding pressure time (x_2), and cooling time in the mold after the filling (x_3). The control parameters with their boundaries are listed in Table 2.2. The objectives, estimated by Cadmould 3D-F in each simulation run used in this thesis, are given in Table 2.3. Typically, the optimization problem would be addressed using multi-objective optimization. However, in this section we focus on single-objective analysis to evaluate the three test instance generation strategies. A multi-objective approach is presented in Sec 6.2.

Table 2.2: *Chosen injection molding simulation control parameters*

Name	Range	Description
x_1	$[230, 250]$	melt temperature [°C]
x_2	$[0.3, 6.3]$	holding pressure time [s]
x_3	$[10, 50]$	cooling time in the mold after the filling [s]

Table 2.3: *Injection molding simulation objectives*

Name	Objective	Description
y_1	minimize	deviation from target mass value [g]
y_2	minimize	max. volume shrinkage [%]
y_3	minimize	avg. volume shrinkage [%]
y_4	minimize	required cooling time [s]
y_5	minimize	max. warpage [mm]
y_6	minimize	max. deformation [mm]
y_7	minimize	avg. shrinkage [mm]

2.3.2 Experimental Setup

We fit the *level 0 model*, which serves as the ground truth for further comparisons, using Cadmould 3D-F evaluations with three different sizes (10, 15, 20) of Latin Hypercube Design (LHD) of the parameter search space (see Table 2.2) and three repetitions each to account for the randomness in the LHDs. The test instances regarding the objectives (y_1 to y_7) delivered by the IM simulation (see Table 2.3) are fitted using a single GPM each. Because three replicates are run for each design size, nine ground truth test instances are generated for each objective. The 1-stage and 2-stage simulation variants are parameterized as follows. A regular grid was used as the design to sample the desired number of test instances (5) for each unique configuration by varying the $Ncos$ parameter from the range $[50, 200]$ with 50 different values and the $conditional$ parameter with the values true and false. This results in 500 *level 1 models* (5x50x2) per objective as test instances for the 1-stage variant and 500 test instances per objective for the 2-stage variant. The PV was parameterized as follows. A regular grid of five points is sampled for each variation parameter (λ, θ_i, $i = 1..4$), resulting in a total of 625 instances per objective (5^4).

Baseline comparison To evaluate the retrieved test instances, we compare them with the artificial test functions from the BBOB test set. We use the function ids (fid) 1-24 and instance ids (iid) 1-10 and set the number of dimensions to three, equivalent to our IM simulation problems, and the parameter search space for each dimension to $[-5, 5]$. This leads to a total of 240 BBOB test instances.

For each test instance, a set of ELA features will be computed using Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems (flacco), a software package for the R platform. Following [85], we compute non-correlated features invariant to rotation and shifting. This is useful to a) decrease the dimensionality by focusing on relevant features and b) remove misleading features, as the GPS also may shift and rotate the test instance. The resulting list of ELA features after removing redundant features is shown in Table 2.4. To compute the selected features for each test instance, flacco was configured to sample them with LHDs of size 1500.

Elo developed its rating system to calculate the relative skill levels of players in chess [49]. Adapted to algorithm performance, each competing algorithm is considered a player, and the test instance is the game. After a certain amount of test instance evaluations, pairwise comparisons of the algorithm's results determine the winning algorithm, i.e., the algorithm that found the lowest objective function value. After every game, the winning player takes points from the losing one, and the number of points is determined by the difference in the two player's ratings.

We will use Elo scores to analyze the impact of the test instance generation strategies and their parameters on the performance of algorithms. This must be processed to ensure, that benchmarks for algorithm selection based on generated test instances are valid. Consequently, we only evaluate the generated test instances, and not the

Table 2.4: *Selected landscape features for the test instance validation*

Feature names 1-8	Feature names 9-16
cm_angle.angle.mean	ela_meta.quad_w_interact.adj_r2
ela_distr.skewness	ela_meta.quad_simple.adj_r2
ela_distr.kurtosis	ela_meta.lin_w_interact.adj_r2
ela_distr.number_of_peaks	disp.ratio_mean_02
ela_meta.lin_simple.adj_r2	disp.ratio_median_25
ela_meta.lin_simple.intercept	pca.expl_var_PC1.cov_init
ela_meta.lin_simple.coef.min	pca.expl_var.cor_init
ela_meta.lin_simple.coef.min	pca.expl_var.cor_init

Table 2.5: *Selected algorithms and their control parameters*

Algorithm	Package (version)	Parameter	Value
GenSA	GenSA (1.1.7)	acceptance_param	-5
		visiting_param	2.62
		temperature	5230
DE	DEoptim (2.2-6)	NP	30
		F	0.8
		CR	0.9
		strategy	2
GENOUD	rgenoud (5.8-3.0)	pop.size	30
BOBYQA	minqa (1.2.4)	-	-
COBYLA	nloptr (1.2.2.2)	-	-
RS	-	-	-

BBOB functions. We set the number of repeats for each algorithm on each test instance to 500, i.e., the Elo rating is computed after 500 runs of each algorithm, with an initial Elo score of 1000. The maximum number of function evaluations (i.e., the algorithm budget) is 300. We chose several different algorithms as follows: Generalized Simulated Annealing (GenSA) [151], Differential Evolution (DE) [108], Genetic Optimization Using Derivatives (GENOUD) [72], Bound Optimization BY Quadratic Approximation (BOBYQA) [126], Constrained Optimization BY Linear Approximations (COBYLA) [125], and uniform Random Search (RS) as a baseline comparator. All algorithms with their package, version, and parameter configurations are listed in Table 2.5. For parameters that are not listed, we applied default values.

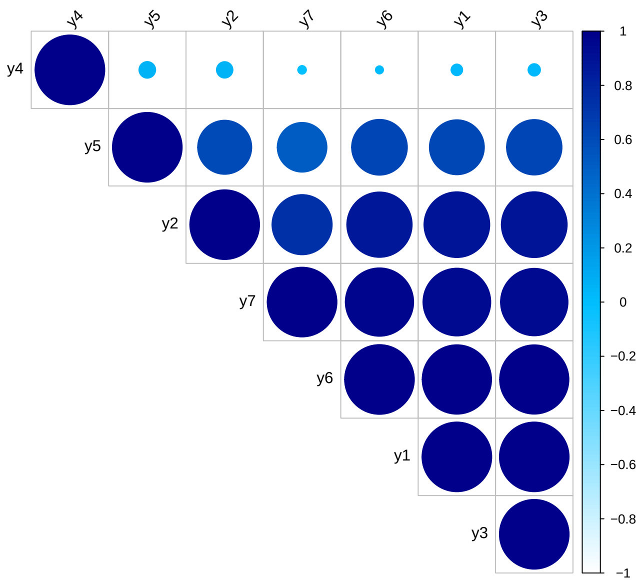

Figure 2.6: *Pairwise correlations using Pearson correlation coefficient between the objectives y_1 to y_7. The objectives y_2, y_7, y_1, and y_3 are highly positively correlated. Objective y_4 do not show a high correlation, and y_5 show some mild correlation to the other objectives. This Figure is based on [52, p. 7].*

2.3.3 Results

To reduce the complexity of the analysis, we aim at dimensionality reduction with respect to the objectives. The training data for the *level 0 model* is analyzed for the degree of linear correlation between the different objectives, with the goal of selecting largely uncorrelated objectives for further investigation. The upper triangular matrix showing the Pearson correlation coefficients of all initial designs (design sizes 10, 15, and 20 and 3 replicates each) is shown in Fig. 2.6. The first objective, y_1, the deviation from the target mass, is highly correlated to the objectives y_2, y_7, y_6, and y_3. This is intuitively explainable, as finding a parameter setup that reaches the target mass will minimize objectives like volume shrinkage, warping, and deformation and vice versa. The objectives y_4 and y_5 are not highly correlated to all other objectives. Additionally, the objectives y_2, y_7, y_6, and y_3 are highly positively correlated. As a consequence, we choose y_3 as a representative objective for them. Objective y_4, the required cooling time, addresses the process's cycle time besides the quality of the products. This results in the following largely uncorrelated objectives to minimize, which will be analyzed further: y_3 (avg volume shrinkage), y_4 (required cooling time), and y_5 (max. warpage).

The computed ELA features will be visualized two-dimensionally using t-distributed Stochastic Neighbor Embedding (t-SNE) [4]. T-SNE tries to preserve local neighborhood information when projecting the data to lower dimensions. The first visual analysis of the feature space covered by the generated test instances is depicted in

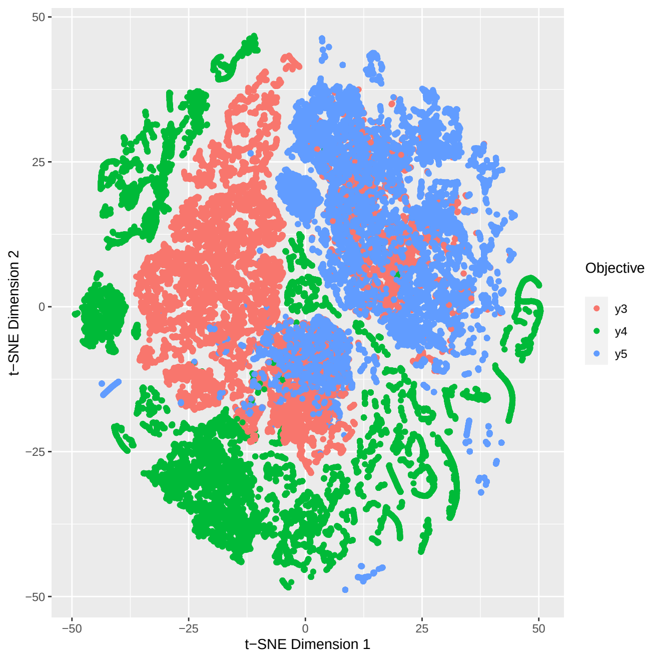

Figure 2.7: *T-SNE visualization of the feature space of all computed test instances based on different design sizes (10, 15, 20) and selected objectives (y_3, y_4, y_5). A tendency of cluster formation based on the objective can be recognized, despite some overlap of the objectives y_3 and y_4.*

Fig. 2.7. Regardless of the generation method, the test instances based on the different objectives form larger clusters per objective. Especially the objective y_4 (required cooling time) does not show much overlap with the other two selected objectives, while y_3 (avg. volume shrinkage) and y_5 (max. warping) share at least a portion of test instances in common clusters. We conclude that the different objectives lead to differentiable problem classes, even if some test instances of different objectives share some ELA-feature-based similarity.

In the next step, we will compare the different test instance generation methods with test instances from the BBOB test set. Consequently, we will visualize the test instances per objective and restrict to the expectedly most accurate generation method based on the LHDs consisting of 20 points. Fig. 2.8 shows the feature space visualization per objective. For each objective, it can be seen that the generated PV and GPS instances differ from the BBOB instances. Additionally, the 1-stage and 2-stage simulation-based instances show some similarity, which could already be observed in the previous Fig. 2.7. As the 2-stage simulation can be informally interpreted as added minor ruggedness to the 1-stage simulation, a fundamental similarity is already expected. The ground truth instances are the model estimates of the *level 0 models*, upon which the test instances are built. It can be seen that the test instances differ from the ground truth, which shows the diversity of the generated problem instances, i.e., the diversity inside the class.

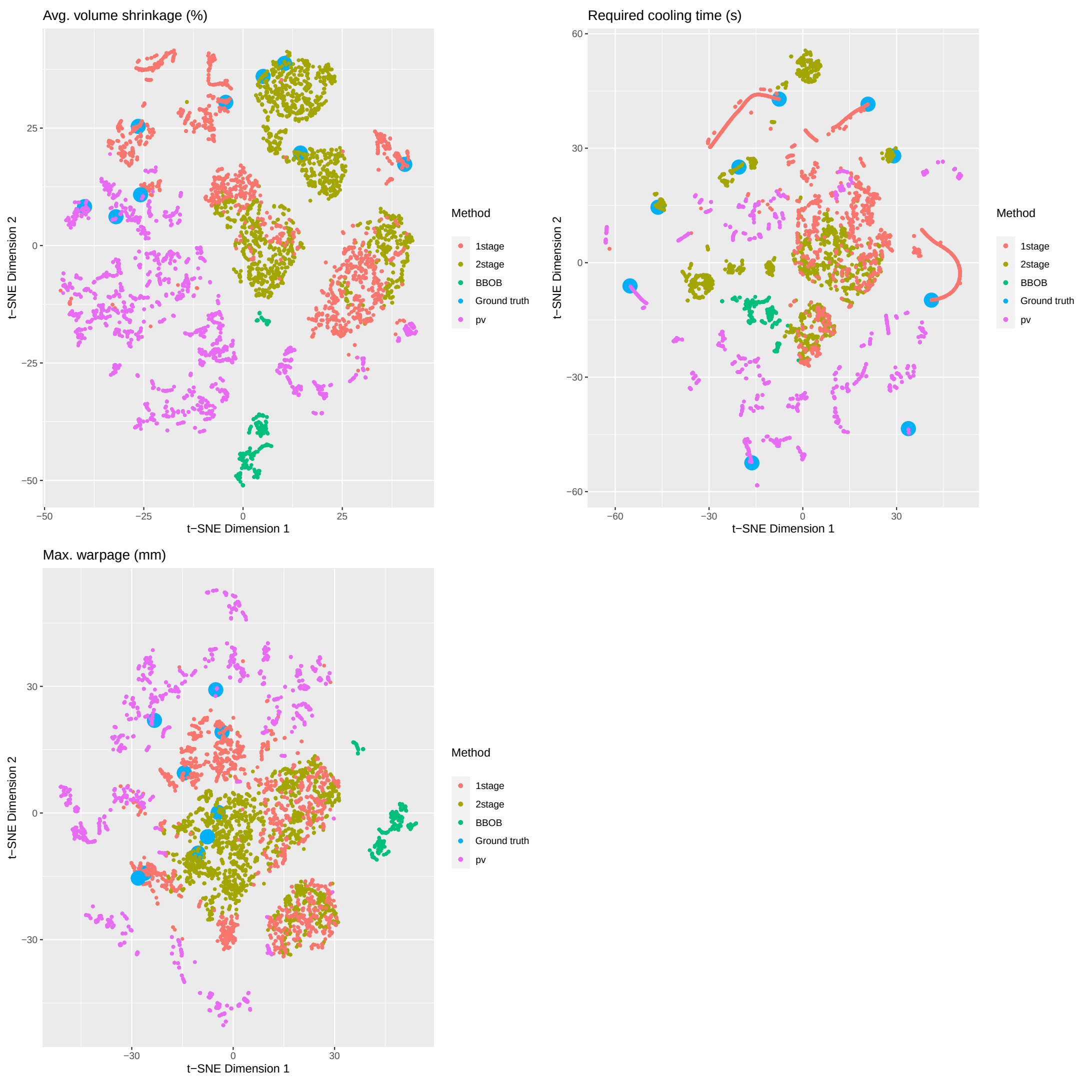

Figure 2.8: *t-SNE visualization for each objective with respect to the test instance generation method related to the BBOB instances and the ground truth, which is the model estimate upon which the test instances are built. The upper left figure shows the objective $y3$ (avg. volume shrinkage), the upper right depicts the objective $y4$ (required cooling time), and lower left the objective $y5$ (max. warpage).*

In conclusion, the BBOB instances are not relevant for the single-objective optimization problems of injection molding simulation. Furthermore, the generated test instances provide a certain amount of variety. This ensures that we do not generate the same or too similar instances. This directly answers to **RQ-1.1**, that the presented strategies are feasible to generate test instances that are similar to the ground truth and provide the desired variety.

Next, we want to inspect the impact of parameters of the test instance generation on the problem class. For the 1-stage and 2-stage simulations, we will look at both the

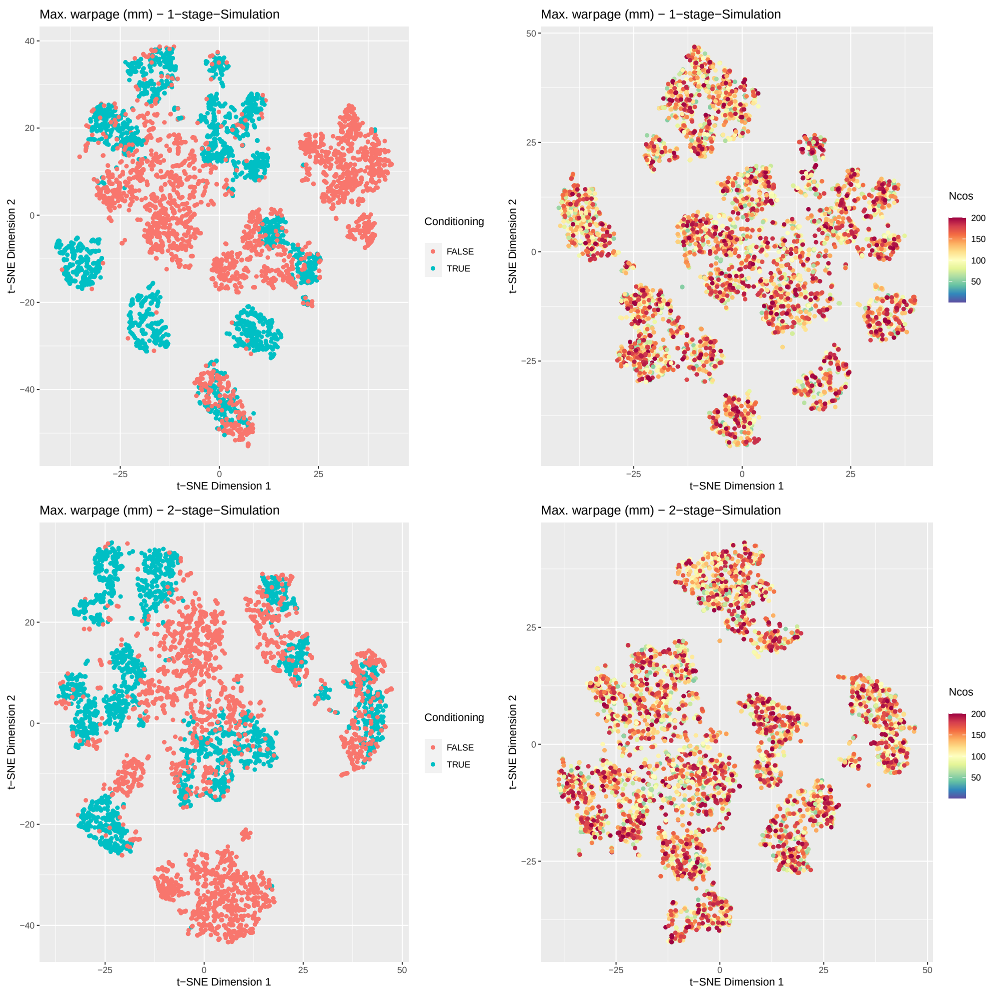

Figure 2.9: *T-SNE visualization for 1-stage and 2-stage simulation for objective $y5$ (max. warpage) with respect to the parameterization. The first row depicts the impact of the conditioning (left) and $Ncos$ (right) on the problem class using 1-stage simulation, and the second row depicts the 2-stage simulation-based problem class.*

conditioning to the training data and the impact of the parameter $Ncos$. We select the objective y_5 (max. warpage) as a representative, as all three objectives show similar behavior. The results are shown in Fig. 2.9. We observe that the conditioning forms recognizable clusters for both 1-stage and 2-stage simulations. Regarding the ELA-features, the parameter conditioning might lead to largely separatable problem classes. When looking at the number of cosines ($Ncos$), changing values only slightly changes the positioning of the instance in the feature space.

The evaluation of the impact of the parameters of the PV method, exemplarily depicted in Fig. 2.10, reveals that the impact of the α value on the instance space positioning depends on the value or importance of the corresponding λ or θ value. On the first row, we see the instance space for the objective y_3 (avg. volume shrinkage): The left side shows the impact of the change on λ, and the right side shows the impact

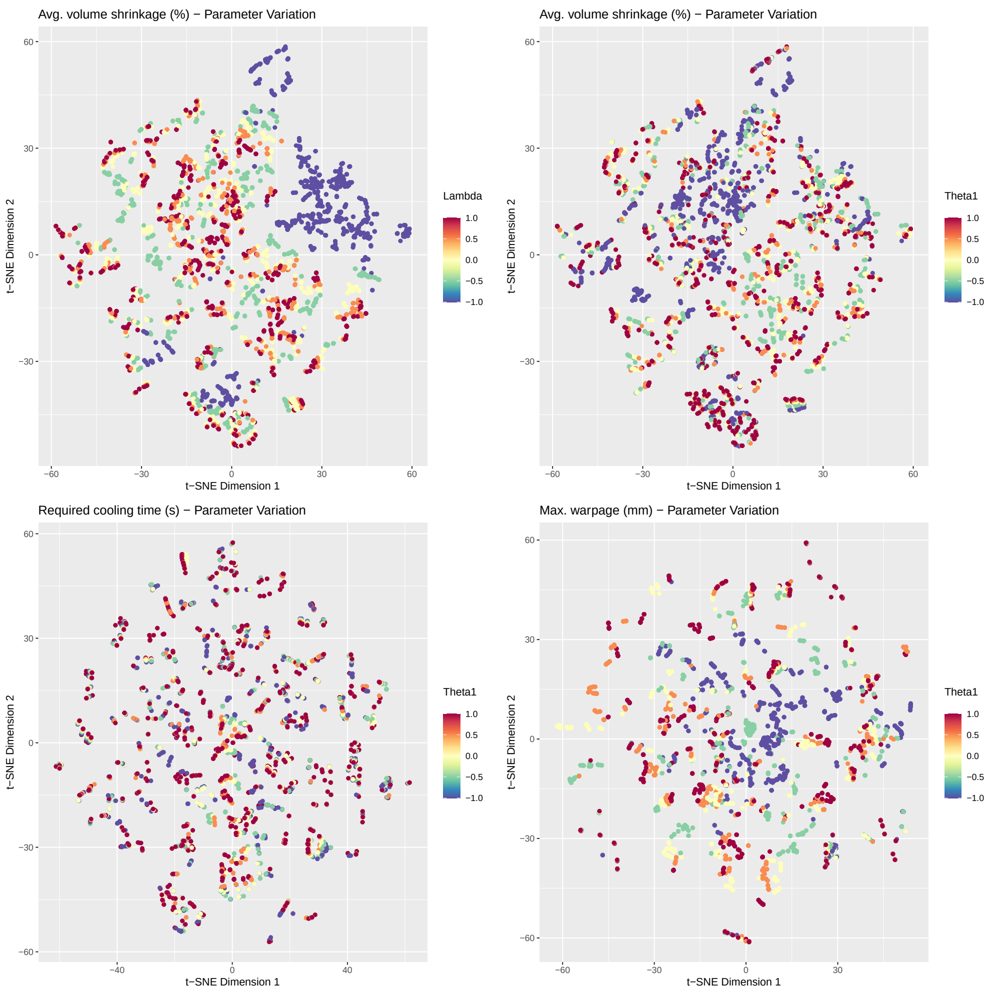

Figure 2.10: *Impact of the parameter variation of λ (upper left) on objective y_3 (avg. volume shrinkage) and θ_1 on the objectives y_3 (upper right), y_4 (required cooling time, lower left), and y_5 (max. warpage, lower right).*

of the change on θ_1, which corresponds to the control parameter x_1 (melt temperature). Instances with either high (1.0) or low (-1.0) values on a normalized scale (range $[-1...1]$) for changes in λ are placed recognizable close to each other. Accordingly, on the right side, we can observe spaces with rather large amounts of instances gathered together with either high (1.0) or low (-1.0) values for the variation of θ_1. This can be explained as, for the objective y_3, the control parameter melt temperature has a strong influence. Contrarily, the lower left shows an example where the value of a parameter (in this case θ_1) does not have a huge impact on the formation of clusters for the objective y_4 (required cooling time). The corresponding control parameter melt temperature does not influence the required cooling time. On the lower right, we can again observe the impact of the extreme variation of θ_1 for the sensitive parameter melt temperature on the objective y_5 (max. warpage).

To summarize the feature space visualization results, we can see that the GPM

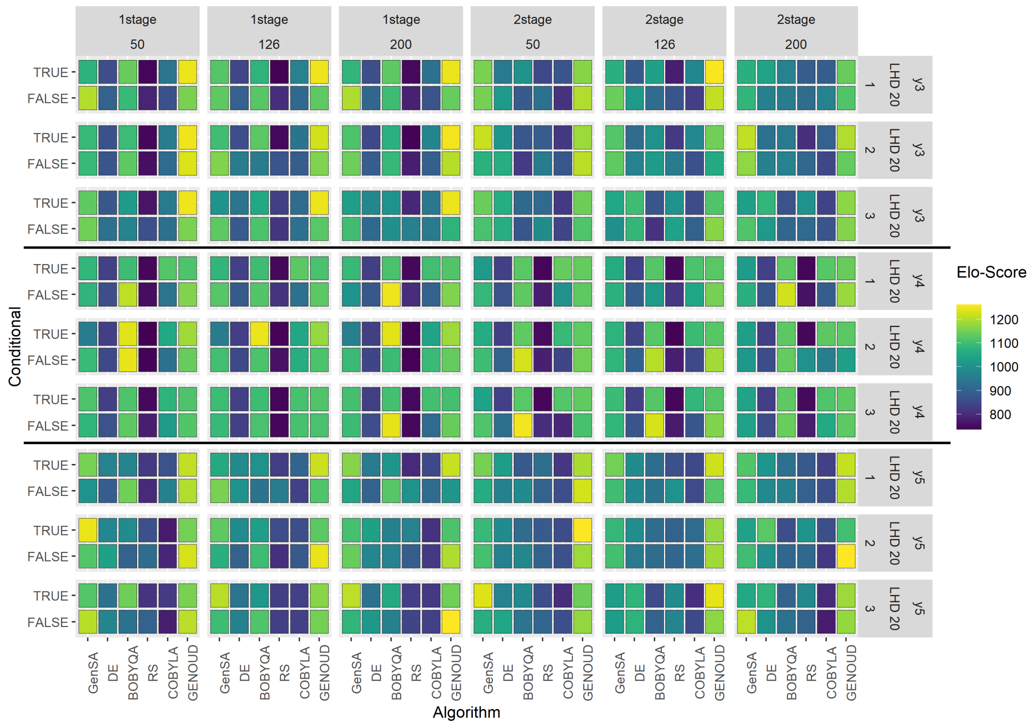

Figure 2.11: *Elo-scores of all selected algorithms on test instances generated with GPS for different objectives (separated by the horizontal bars), repetitions, and given configurations shown on the row and column labels. The y-axis depicts the conditioning parameter (left) and the objective, used sampling design, and the design repetition number (all on the right side). The x-axis depicts the method (1-stage, 2-stage), the parameter Ncos (both on the top), and the algorithm (bottom). A higher Elo value indicates a better relative performance.*

based test instance generation lead to recognizable problem classes for each selected objective. Most importantly, the selected ELA features, which have proven to be robust for classifying instances to BBOB problem classes, reveal differences between generated test instances and the BBOB problems, which directly answers (Q-1).

The following results of the benchmark experiment will examine if there is also a difference in the resulting ranks of the algorithm's performance between PV, 1-stage, and 2-stage instances.

We analyze the relative performance of the selected algorithms on one randomly selected test instance per configuration, i.e., the objective (y_3-y_5), the simulation method (PV, 1-stage, 2-stage) and the corresponding parameter vector $\vec{\alpha}$, i.e., the $conditioning$ ($true/false$), the $Ncos$ (50-200), or the λ and θ_i changes using Elo-scores. We are especially interested in changes in the algorithm's ranking on the objectives by varying simulation or variation parameters, i.e., answering the question if these parameters are changing the difficulty for some of the algorithms.

At first, we look at the Elo scores for the GPS instances depicted in Fig. 2.11. Regarding the first objective, y_3, GENOUD is often the superior algorithm with the highest Elo score, even if its performance dropped slightly for the 2-stage variants. Objective y_4 leads to a different rank order, where BOBYQA most often performs best. For objective y_5, GenSA and GENOUD mostly outperform the other algorithms. Regardless of the changes in the test generation or the design repetition, the rank order is preserved. Mainly, the objective determines the best-suited algorithm. COBYLA only performs competitively on objective y_4, mainly on the conditioned variants. This is intuitively explainable, as this objective, the required cooling time, is mainly linear dependent on both the cooling time in the mold and the holding pressure time. The global optimizers (DE, GenSA, GENOUD) perform especially well on the more complex objectives y_3, which is relatively smooth, and y_5, which has a larger degree of multi-modality.

The analysis of the Elo scores for the test instances generated with parameter variation are consistent with the previous results. The performance ranks are largely preserved among the different parameter values of $\vec{\alpha}$ for each objective and repetition.

To summarize the results of the performance of the algorithms, we can say that regardless of the test instance generation method, the performance ranks of the algorithms on the selected instances are kept mainly for each objective, which answers **RQ-1.2**. While conditioning affects the Elo score itself, not so much on the performance ranks, we recommend applying conditioned GPS instances when an estimate of the objective values is needed, or the structure of the problems should be kept for, e.g., multi-objective optimization.

2.4 Conclusions

In this section, we specified the problem taxonomy used in this thesis and discussed three different strategies for GPM-based problem class generation for CPPS. We evaluated the generation of test instances for three different injection molding problem classes, namely average volume shrinkage, required cooling time, and maximum warpage. For the evaluation, we emphasized both, numerical ELA features of the test instances and algorithm performance.

We have shown that the problem classes generated with GPS and PV based on process data differ significantly from the problem classes of the well-known and widely used BBOB test set in terms of ELA feature vectors. Therefore, the BBOB test functions are not relevant for our use case, which answers subquestion **RQ-1.1**.

Furthermore, the ranking of the algorithms in terms of their performance is largely preserved, i.e. the test generation parameters do not significantly affect the relative performance of the algorithms as represented by their Elo score. The PV strategy tends to smooth the data, which is not easily counteracted by its parameterization. This is not the case with GPS. Therefore, we will use 1-stage conditional simulation as the default method. Regardless of the method used to generate the test instances, the performance ranks of the algorithms on the selected instances are primarily maintained for each objective , which answers question **RQ-1.2**.

However, when the dimensionality of the optimization problem becomes rather large, the computation of a GPM may consume too much CPU time, depending on the CPPS and the available computational resources. In a modular implementation, the test instance generator should therefore be replaceable, e.g. by a less computationally expensive solution, which may come at the cost of less accuracy.

Chapter 3

Algorithm Classes and Performance Metrics

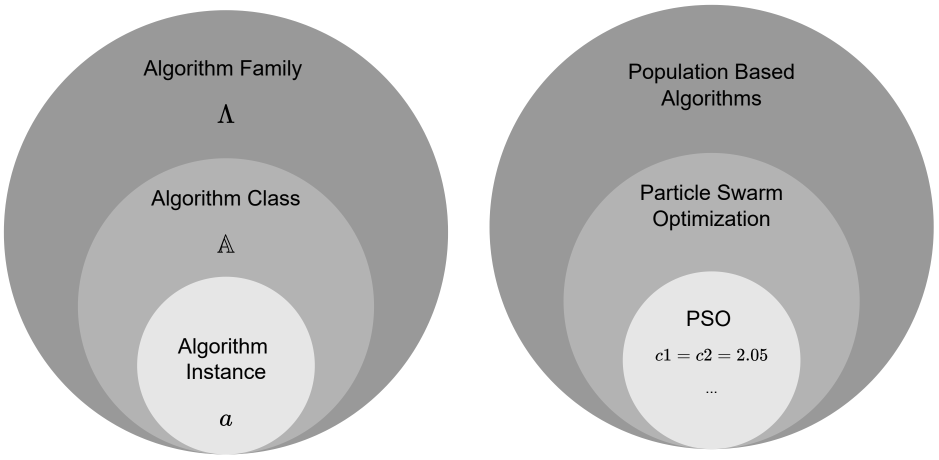

Figure 3.1: *Schema of the Algorithm Taxonomy used in this thesis. The general hierarchy is given on the left side, and an example is shown on the right side. An algorithm instance represents an algorithm with a chosen parameterization. In the best case, it is tuned to the problem class to solve. The Figure is based on [9].*

3.1 Algorithm Classes

Parts of this Section are based on the publication [9], where the taxonomy was first published. The description of the algorithms and the algorithm families was sharpened for the application of this thesis. Parts of the text were taken verbatim.

In the previous section, a classification scheme for optimization problems was provided. During the evaluation of problem classes created with the test instance generator based on production process data, it was already pointed out that different problem classes favor different algorithms and vice versa. Therefore, this section deals with a possible way to describe and classify algorithms in order to select an algorithm portfolio that is able to solve several problem classes.

To assess the quality of different algorithmic ideas, comparing algorithm instances from different families is helpful. An online solution for the ASP will likely only have some feasible algorithms available in an implementation. Still, it can be crucial to have different algorithms at hand, depending on the dimensionality and the type of the optimization problem at hand. The taxonomy described in this section aims to have algorithms from different families which address likely different features of problems. This can be features stemming from the problem definition, e.g., the dimensionality of the search space and the available budget, as well as numerical features from the fitness landscape, i.e., features computed via ELA.

The presented classification covers a large part of common optimization heuristics. A common feature of all of them is a high degree of parameterization of the algorithms, which requires hyperparameter optimization (tuning) on the given problem class to be solved in order to reveal their full performance potential.

Extending the work of [144], we discuss five different families of algorithms:

(a) One-Shot Optimization algorithms

(b) Monotonic descending algorithms

(c) Non-monotonic descending algorithms

(d) Population-based algorithms

(e) Surrogate-based algorithms

Arguably these families are given in an order with increasing complexity.

One-Shot Optimization Algorithms When dealing with problems that are very costly to evaluate or when simply working under high time pressure, sequential optimization may not be possible. In such cases, the decision maker has to resort to *one-shot optimization algorithms,* which typically sample the decision space in a space-filling manner. Several criteria to measure the "space-fillingness" exist, and each one implies different one-shot designs. The debate about which variant to favor under which circumstances is closely related to a similar discussion around the Design of Experiments (DOE). One-shot algorithms are typically used to determine an initial set of candidate solutions in population-based or surrogate-based algorithms.

Quasi-random designs such as LHD and low-discrepancy point sets [42, 98] are often found to be superior over uniform sampling, see [27, 21, 14] for related theoretical works. It is worth noting that even when the prior distribution of the optimum is exactly known, the best one-shot distribution may be a different one [103] - a phenomenon related to the "Stein phenomenon" [143, 71].

We also classify as one-shot optimization algorithms those strategies in which the final decision may differ from the evaluated points. In this setting, the user evaluates n alternatives but is free to use the information these n points provide to decide on an alternative that is not included in this set. The final decision can be derived by optimizing a surrogate built on top of the n points [20] or by simply averaging some of the best points [102].

Thus, summarizing this paragraph, the distinguishing property of one-shot optimization algorithms is that no adaptive sampling is permitted, i.e., all points to be evaluated by the algorithm have to be decided on independently of the quality of other search points.

Monotonic Descending Algorithms Besides deterministic or stochastic algorithms, this family covers gradient-based algorithms. The concept of exploration is often not part of these algorithms, which means that escaping local optima is likely not possible.

Consequently, monotonic descending algorithms (or hill-climbers) are often embedded in more sophisticated global search strategies to enable fast convergence in a local optimum. Examples for these algorithms include the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [137], conjugate gradients [55], and Nelder-Mead [117]. Hill-climbers typically start with a single solution and employ a greedy search strategy. The performance of these algorithms relies heavily on the starting point, which is, due to the absence of individual operators, selected at random or by utilizing domain knowledge. Therefore, restarts with different (random or quasi-random) starting points are a common approach to using hill-climbers effectively. In comparison to the range of the search space, new candidate solutions are searched within the vicinity of the current solution. Gradient-based methods compute or approximate the gradients of the objective function to find the potentially best direction and step size for the improvement of the solution. The step size is the most critical control parameter, as it controls convergence speed. Using an adaptive strategy to change the step size online is considered state of the art nowadays, e.g., with the well-known $1/5$-th success rule, see [129, 136, 41].

Non-Monotonic Descending Algorithms Algorithms of this family extend the idea of hill-climbing algorithms, so they are aware that multiple places of interest exist. Exploration is added as a concept of this family, and the search-space is systematically evaluated by one or more candidate solutions. Algorithms of this family consider operators to guide the search globally in specific directions. A well-known for this family is Simulated Annealing (SA) [81]. SA implements an acceptance function that allows the escape of local optima with a certain probability, i.e., the acceptance of a worse solution, which enables exploration to a certain degree. Modern SA variants implement self-adaptive acceptance functions, which allows, e.g., alternating phases of exploitation and exploration.

Population-based Algorithms Population-based algorithms utilize several candidate solutions concurrently. This makes them especially useful when the objective function can be evaluated in parallel. Examples are Particle Swarm Optimization (PSO) [79, 138], Ant colony optimization for continuous domains ($ACO_{\mathbb{R}}$) [44, 142], and several Evolutionary Algorithms (EAs). EAs can arguably be considered the state of the art of the population-based algorithm family, as their search concepts are fundamental for this field of algorithms. EAs use paradigms from natural evolution, such as selection, recombination, and mutation, to guide a population of individuals over several generations toward optimal or near-optimal solutions [6]. EAs generally have several tunable parameters, e.g., the selection pressure, variation step size, mutation, and recombination rate. For an overview of the field of EAs, we refer to [6, 48]. For an overview of existing methods to tune EAs, we refer to [47, 67].

Surrogate-based Algorithms Surrogate Model-Based Optimization (SMBO) distinguishes two subfamilies of algorithms using surrogates. Surrogate-based algorithms, which utilize a surrogate model of the objective function for variation and selection, and Surrogate-assisted algorithms, which employ surrogate models to improve the selection of candidate solutions in population-based algorithms. For each group, the quality of the surrogate model is crucial for the efficiency of the algorithm. Assuming that different modeling techniques have different advantages and disadvantages, it is difficult to choose the best-fitting model. Popular examples of surrogate model techniques are regression models (linear, quadratic, polynomial), GPM, regression trees, artificial neural networks, and symbolic regression models. Recent research focuses on ensembles of surrogate models to exploit the advantages of different models simultaneously to compute predictors superior to a (or any) single predictor. A well-known example of surrogate-based optimization is the Efficient Global Optimization (EGO) algorithm, see [76]. Several control parameters are introduced, e.g., the size and sampling strategy used to generate the initial solutions, the employed model technique, the infill criterion for the candidate selection, the optimizer that searches the surrogate for these candidate solutions, and the adaptation technique to converge from an exploratory infill criterion to an exploitative one. As surrogates are often considered cheaper (in terms of evaluation time or material cost) compared to the objective function, they are often (but not exclusively) employed in low-budget scenarios, where only a few evaluations are affordable, e.g., the parameter optimization utilizing complex CFD simulations or hyper-parameter optimization.

This taxonomy gives an insight into the development of algorithm families over the last decades and a rough idea of the diversity of implemented concepts, which also address different features of optimization problems. As these features are often not known in advance, a benchmark experiment can help to extract knowledge about the strengths and weaknesses of selected algorithms on the given problem. This is only feasible when relevant test instances are available or - as this thesis suggests - can be generated. Typically, some expert knowledge can be applied to preselect a group of potentially efficient algorithms, apply hyperparameter optimization (tuning), and select the best-performing algorithm instance. This knowledge might not be available in the described scenario of this thesis, which needs automatic online algorithm selection. However, this implies that an even more diverse algorithm portfolio is needed to cover a broader set of problems. The goal is to select the best-performing algorithm out of the defined portfolio, contrary to the best-performing in theory. Consequently, this might exclude not only bad-performing algorithms but also good or even best-performers when resource (i.e., time or computational resources) constraints still need to be met. This needs the employment of performance metrics to compare and finally select an algorithm. These metrics are discussed in the following section.

3.2 Performance Assessment

This section discusses the evaluation of algorithms' performances on single or multiple optimization problems. Parts of this section are based on the publication [51], which demonstrates an ANOVA-based performance evaluation approach that allows for generalization to a problem class by taking the randomness of the test problem selection into consideration for the statistical analysis.

This thesis aims to develop and implement a cognitive system to help optimize CPPSs mostly automatically. It will be cognitive in the sense of utilizing sensor data to increase the efficiency of the production process. The main feature is not to procedurally program the solution but to let the system learn to apply (optimization) methods. This can be made possible by allowing the system to assess its performance numerically and thus let it compare the outcome of different methods. This section covers the most important available performance measures for continuous single-objective optimization algorithms and ways to compare the results of several algorithms.

When benchmarks are performed, most of the time one of the following two questions needs to be answered [8]:

(Q-1) How quickly can a given quality of solution be achieved?

(Q-2) What quality of solution can be achieved with a given budget?

They are also of particular importance for the operator of a CPPS, e.g. after an adaptation of the production process, when a parameter optimization is required. When comparing the performance of different algorithms on an optimization problem instance, we can distinguish between a fixed target (Q-1) and a fixed budget (Q-2) approach , as shown in Fig. 3.2.

Fixed Target: A certain objective function value to be achieved is specified, and the number of function evaluations to achieve this target is measured for each algorithm. A fixed target can be represented as a horizontal line, see Fig. 3.2.

Fixed Budget: The maximum available budget in terms of objective function evaluations is given, and the achieved objective function value of the algorithms is measured. The budget can be represented by a vertical line, see Fig. 3.2.

We can then choose the algorithm that reaches a fixed goal first, or the algorithm that reaches the best (minimum) objective function value after a given number of function evaluations. In the context of CPPS, the target value to achieve is sometimes not known in advance, especially after an adaption of the process or when considering energy consumption minimization.

We aim for results that can be substantiated by statistical tests that compare several repetitions of the selected algorithms. These tests should answer the following questions:

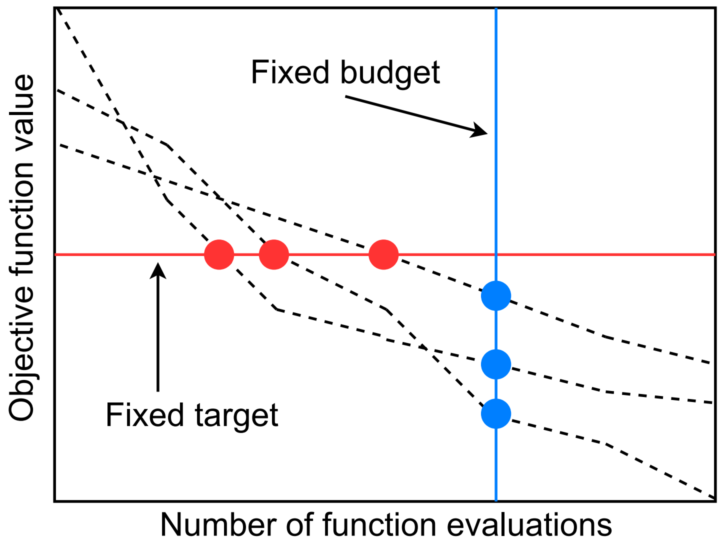

Figure 3.2: *Visualization of a comparison of achieved (minimum) objective function values by three different optimizers (dashed lines) using a fixed-budget (blue line) and a fixed-target (red line) approach. The fixed-budget scenario requires a defined number of function evaluations at which the best achieved results are compared (vertical line), and the fixed-target scenario requires a function value to be specified a priori (horizontal line). The Figure is based on [62, p.5].*

(Q-1) Are there significant differences between the results of the different algorithms?

(Q-2) If so, which ones are different?

(Q-3) Which algorithm is superior?

If we are evaluating performance on multiple test instances, we need a test that takes into account blocked data, where each block represents the problem instance. Considering the available tests, two groups of relevance can be distinguished: Parametric and non-parametric tests.

Parametric tests are those that make assumptions about the parameters of the distribution from which the samples are drawn. Analysis of Variance (ANOVA) [106] is a well-known example of a parametric test. Often, independence, normality, and homoscedasticity of the data or the residuals of the corresponding model are assumed. Independence means that the result of a single experiment, i.e., one run of an algorithm, does not affect the result of another experiment. Normality means that the data or the residuals of the corresponding models are normally distributed. Homoscedasticity, or homogeneity of variance, means that the variances in different groups (in our case, groups are the different algorithms) are similar or equal.

At first, we discuss an approach for parametric tests based on ANOVA, which can be applied if the requirements are met. For the case, that the assumptions for

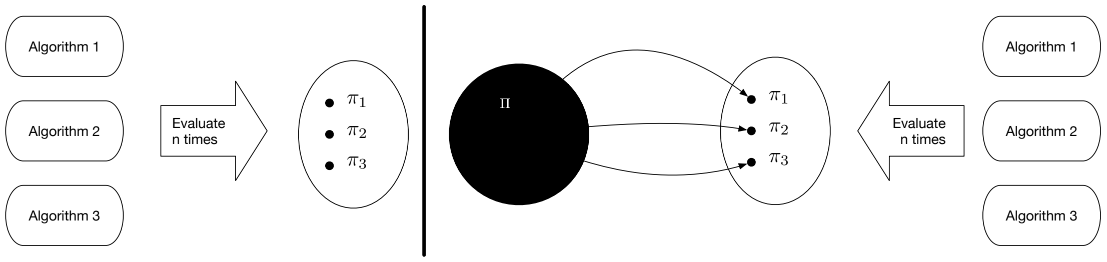

Figure 3.3: *Scenarios for selecting a number of test instances and evaluating a number of optimization algorithms performances. In this example, the number of problem instances, n_π, and the number of algorithms, n_a, are set to three.* Left: *A fixed set of test instances is previously selected. Problem class information is not used, and results can not be generalized.* Right: *Test instances are randomly selected from problem class Π and are not known in advance, results can be generalized, and propositions of the performances of the algorithms on the whole test class can be made. The problem class can be regarded as a black box. The Figure is taken from [51, p.2]*

the safe use of parametric tests do not hold, which is quite possible [40], we discuss non-parametric tests.

Parametric Tests Statistical tests are performed to analyze the effect of a factor of an experiment. A factor is an independent variable, such as an algorithm or a test instance in a benchmark experiment, and the factor levels represent values for the factor. Based on the ideas of Chiarandini and Goegebeur [29], we propose the following approach: A small set of problem instances $\{\pi_i \in \Pi | i = 1, 2, \ldots, n_\pi\}$ is randomly chosen from a larger class Π of possible instances of the problem. The problem instances π_i are called factor levels. This factor is different from the fixed algorithmic factors in the classical ANOVA setup. In fact, the levels are chosen randomly, and the interest is not in these specific levels, but in the problem class Π from which they are sampled. Therefore, the levels and factors are *random*. In contrast to the classical approach, which is illustrated in the left panel of Fig. 3.3, our results are not based on a limited, fixed number of problem instances because they are randomly drawn from an infinite set as illustrated in the right panel of Fig. 3.3. This approach avoids overfitting and enables generalization.

Bartz-Beielstein and Preuss [10] introduced a classification schema for benchmarking that was extended as follows [51]:

- one single algorithm is tested on a single problem instance (SASP),
- **one single algorithm is tested on multiple problems instances (SAMP)**,
- one single algorithm is tested on multiple problem classes (SAMC),
- multiple algorithms are tested on one single problem instance (MASP),
- **multiple algorithms are tested on multiple problem instances (MAMP)**, or
- multiple algorithms are tested on multiple problem classes (MAMC).

We will present results of an ANOVA-based approach that requires basic knowledge of the underlying ANOVA only. The approach is a generic extension of the widely

Table 3.1: *Settings of parameters for generating GLG instances.*

Parameter	Name	Range	Value
d	Dimensionality of the problem instances	$\{2..20\}$	2
nGaussian	Number of single Gauss components to be generated	$\mathbb{N}_+$	10
ratio	Ratio of the values of the local and global optima.	$[0..1]$	0.7
lower	Vector of lower bound values of length d.	$\mathbb{R}^d$	$(-2, ..., -2)^\intercal$
upper	Vector of upper bound values of length d.	$\mathbb{R}^d$	$(2, ..., 2)^\intercal$

used practice of classical ANOVA and hypothesis testing framework. We exemplify the results of such an approach on the SAMP and MAMP scenarios.

Standard models in statistics are based on linear combinations of the factor effects. The standard analysis assumes that the factors remain fixed, i.e., the levels of the factors were the levels of interest. Conclusions from the statistical analysis are valid for these levels. In our setting, we are interested in concluding about a larger population of levels, not only those that were used in the experimental design. Therefore, factor levels are chosen at random from a larger population or class. Fixed factors occur in our models as well: to estimate the effects of algorithm parameters such as population size, the corresponding algorithm factors are systematically changed.

The statistical analysis in this approach is based on models with fixed (algorithm) and random (problem instance) factors, so-called mixed models. For a detailed description of the used models and performed tests we refer to [51].

We use the GLG to generate test instances $\pi_i \in \Pi$ according to the setup given in Tab. 3.1. The GLG places the number of gaussian components randomly inside the user defined lower and upper boundaries. Furthermore, the Generalized Simulated Annealing (GenSA) algorithm was chosen. We consider $n_\pi = 4$ problem instances and $n = 30$ repeats, i.e., 120 observations, with a budget of 100 objective function evaluations each. Furthermore, the tuned parameters of the GenSA algorithm ($temp = 65$, $q_v = 85.65$, $q_a = -2.69$) are used. The parameter tuning was performed using SPOT. See Table 3.2 for the configuration, and Fig. 3.4 for the performance improvement. The box plots indicate that the locations of the performance distributions of the four problem instances differ and that there is almost no variance in problem instance `PInst` 3. A look at the data reveals the unexpected behavior: Mean and standard deviation are close to zero: $\overline{y}_{3.} =$ 4.74e-16 and $\sigma(y_{3.}) =$ 1.80e-15. These values indicate that the third problem instance is not suited for further investigations. We can procede by removing the instance.

The random factor effects model will be used to answer the following questions, which are all related to the robustness of algorithms, in the SAMP setting:

SAMP-1 Does the algorithm a show a different performance on problem instances π_i that were randomly drawn from a problem class Π?

Table 3.2: *SPOT configuration for parameter tuning*

Parameter	Value
spot budget (n_s)	40
design method	LHD
design size	20
model type	Random Forest
optimizer	Nelder-Mead
optimizer budget (n_b)	150

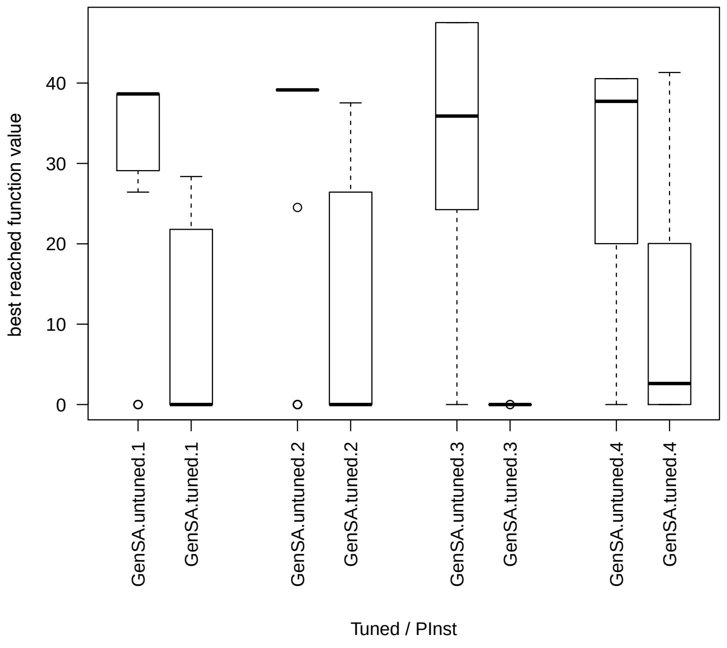

Figure 3.4: *Box plot illustrating the performance of GenSA on four different GLG instances (1-4) tuned and untuned. GenSA.tuned.1 shows the results of the tuned GenSA algorithm on problem instance $\pi_1 \in \Pi$, GenSA.untuned.2 shows the results of the untuned GenSA algorithm on problem instance $\pi_2 \in \Pi$, etc. Y-axis shows the best-reached function value y. Lower values are better.*

SAMP-2 What is the estimated mean performance of the algorithm on this problem class Π?

SAMP-3 What is the 95% confidence interval for this estimated mean performance?

SAMP-4 What is the effect of the extent of variation between the different problem instances?

The null hypothesis of our tests suggests that there is no difference between the

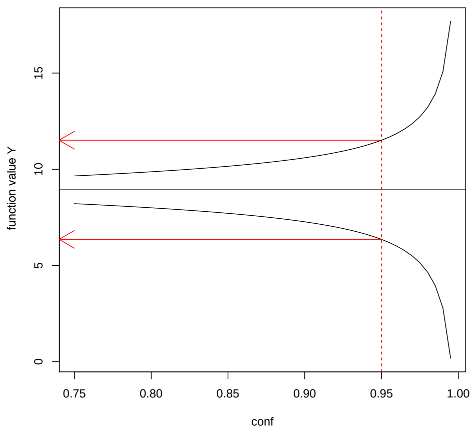

Figure 3.5: *Mean performance of the GenSA algorithm on three randomly selected instances from a pre-defined GLG problem class for given confidence levels from about 0.75 to 1.0. The red arrows indicate the 95% confidence interval [6.36; 11.51].*

compared algorithms or test instances The analysis of variance for this use case reveals that the null-hypothesis can not be rejected. This answers question [SAMP-1]. These results indicate that the mean performance values of the algorithm on different problem instances do not differ. To answer question [SAMP-2] the overall mean value will be estimated, where we obtain $\hat{\mu}. = 8.93$. To answer question [SAMP-3], the 95 percent confidence interval shows that the mean performance of the algorithm is in the interval from 6.36 to 11.51. We can also plot the confidence intervals for varying error probabilities, as shown in Fig. 3.5. Answers to question [SAMP-4] for our example are as follows. Consider, e.g., an error of $\alpha = 10\%$. With a confidence coefficient α, we conclude that the variability of the mean performance values of the different problem instances π_i accounts for between zero and twenty percent of the total variability in the performance values. These answers provides valuable insights into the performance of one specific algorithm on one problem class.

To exemplify the MAMP approach, we chose three algorithms from different families: GenSA, DEoptim, and Sequential Parameter Optimization Toolbox (SPOT). Additionally, a simple uniform random sampling method, RS, is included as a baseline. The test generator parameters and the chosen parameters of the optimizers after the tuning process can be seen in Table 3.3. SPOT was configured to perform the tuning of each method with a budget of 40 runs, and the method's budget was set to 160 evaluations each. The results of the experiments are illustrated in Fig. 3.6. First, we will test for the presence of relevant effects, which results in the following questions:

MAMP-1 Do the algorithms differ in performance?
MAMP-2 Do different problem instances affect the performance?
MAMP-3 Do the algorithms and problem instances interact?

If relevant effects can be detected, then we are interested in their magnitudes. This

Table 3.3: *Resulting parameters for the optimizers and the parameters of the GLG for the MAMP scenario. The budget for each experiment was set to 240 objective function evaluations. The dimension is denoted as d.*

DEoptim	SPOT	GenSA	GLG
popsize=10	designSize=30	temp=65	d=4
itermax=23	designType=LHD	q_v=-15.68	nGaussian=7
strategy=1	modelType=Kriging	q_a=-2.38	ratio=0.6
F=1.81			lower=$(-1, ..., -1)^\mathsf{T}$
CR=0.27			upper=$(1, ..., 1)^\mathsf{T}$
c=0.91			

Table 3.4: *Confidence intervals for the fixed effects contrasts. Significant differences are shown in boldface.*

Algorithm	Difference	d_{ij}	lower 95% CI	upper 95% CI
SPOT - DEoptim	$d_{12} = \overline{y}_{1..} - \overline{y}_{2..}$	-3.17	-14.38	8.04
SPOT - RS	$d_{13} = \overline{y}_{1..} - \overline{y}_{3..}$	-7.99	-19.2	3.22
SPOT - GenSA	$d_{14} = \overline{y}_{1..} - \overline{y}_{4..}$	-23	-33.97	-11.55
DEoptim - RS	$d_{23} = \overline{y}_{2..} - \overline{y}_{3..}$	-4.82	-16.03	6.39
DEoptim - GenSA	$d_{24} = \overline{y}_{2..} - \overline{y}_{4..}$	-19.59	-30.8	-8.38
RS - GenSA	$d_{34} = \overline{y}_{3..} - \overline{y}_{4..}$	-14.77	-25.98	-3.56

results in the following questions:

MAMP-4 What are the fixed factor effects?

MAMP-5 What are confidence intervals for the fixed effects contrasts?

MAMP-6 What are the marginal means?

MAMP-7 How large are the confidence intervals for the marginal means?

A summary of the results answers the questions as follows. To answer [MAMP-1], the analysis reveals that there are significant main effects for factor A (fixed factor, algorithm). To answer [MAMP-2], the analysis reveals there are also significant main effects for factor B (random factor, test instance). For [MAMP-3], the analysis reveals that there are significant factor AB interaction effects as well. Figure 3.7 shows the average performance of the algorithms and illustrates the interactions. For [MAMP-4], the following effects were calculated: $\tau_1 = -8.48$, $\tau_2 = -5.31$, $\tau_3 = -0.49$ and $\tau_4 = 14.28$, where τ_1 is caused by SPOT, τ_2 by DEoptim, τ_3 by RS, and τ_4 by GenSA. For [MAMP-5], the point estimation of fixed effect contrasts for balanced mixed model is shown in Table 3.4. These results can be interpreted as follows: We conclude with a confidence coefficient of 0.95 that SPOT is better than GenSA. Its mean improvement is between 33.97 and 11.55. To answer [MAMP-6], the results of the estimates $\overline{y}_{i..}$ are shown in Table 3.5. To answer [MAMP-7], confidence intervals approximated with the Satterthwait procedure [134] are shown in Fig.3.7. Note that negative values

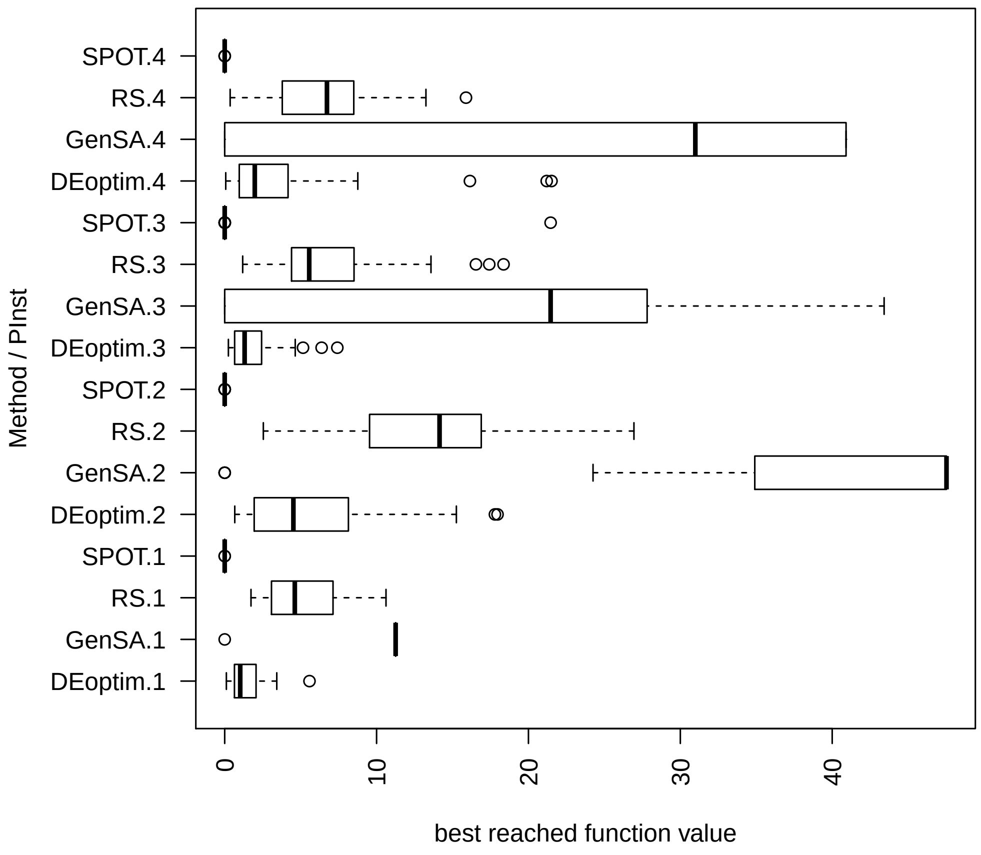

Figure 3.6: *Box plots illustrating the performance of the different algorithms on randomly chosen problem instances $\pi \in \Pi$. Lower y values are better. The global optima have a value of* 0.

occur. These values are a consequence of the definition of confidence intervals (CI). Because the width of the CI increases as the confidence level increases, the CI will cover every region if the confidence level is sufficiently high (and the sample size remains constant). To solve this problem, the confidence level can be decreased, or truncated distributions could be used.

The MAMP analysis can greatly help practitioners to understand the performance of different algorithms on a set of problem instances. The results of these experiments are truly generalizable because they do not depend on a specific (fixed) problem instance, but on a set of several (randomly chosen) problem instances.

When applicable, such a method is a powerful tool for comparing different algorithms and estimating their performance on a (small) class of problems. If the as-

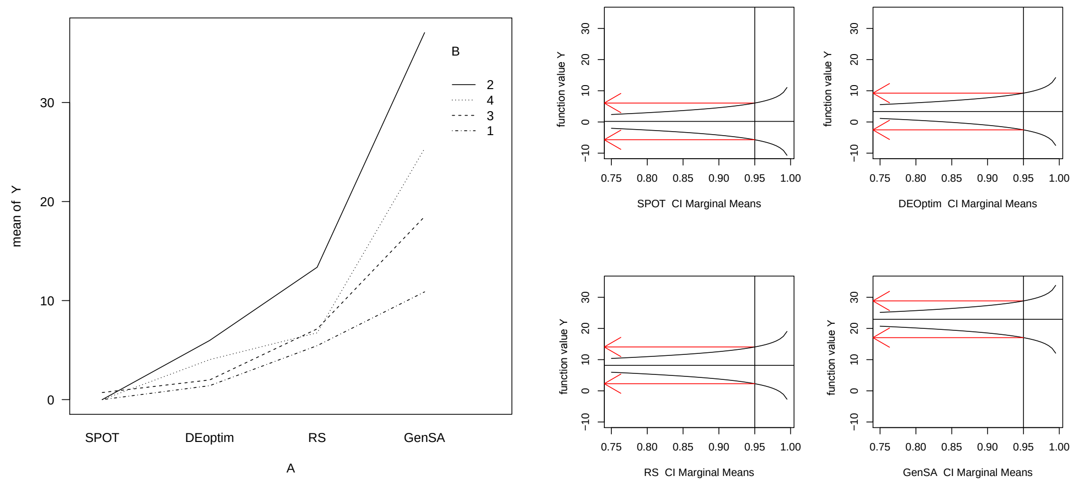

Figure 3.7: Left: *Mean performances on several test functions plotted against algorithms.* Right: *Marginal mean performances of the four algorithms with confidence intervals.*

Table 3.5: *Confidence intervals for the marginal means*

Algorithm	Mean	Value	lower 95% CI	upper 95% CI
SPOT	$\overline{y}_{1..}$	0.18	-5.25	5.61
DEoptim	$\overline{y}_{2..}$	3.35	-2.08	8.78
RS	$\overline{y}_{3..}$	8.17	2.74	13.6
GenSA	$\overline{y}_{4..}$	22.94	17.51	28.37

sumptions for applying a parametric test are not met, or e.g. the performance metric does not allow a meaningful interpretation of the performance estimates, we will apply nonparametric tests as described in the following section.

Non-parametric Tests For the case where we use a single test instance to compare the performance of the algorithms, i.e. the non-blocked case, we first perform a Kruskal-Wallis rank sum test [83] to answer the question whether there are significant differences between the results of the different algorithms (Q-1). The null hypothesis of our tests suggests that there is no difference between the compared algorithms. We reject the null hypothesis if the p-value produced by the test is less than $\alpha = 0.05$. If this test shows that there are significant differences, i.e. the null hypothesis is rejected, a post hoc test for multiple pairwise comparisons is needed. This will answer the question of which algorithms are different (Q-2) [32]. For this test, we will use the multiple pairwise comparisons of Conover [33].

When multiple test instances are used (the blocked case), the same procedure can be applied using the Friedman test and the Conover post hoc test [32, 33]. We will use the implementation from the PMCMRplus package [124] for the R platform.

To answer the question of which algorithm is superior (Q-3), we will use the following procedure, which is also described in [153]. We select the superior algorithm based on the rank sum comparison between the two algorithms under consideration. To obtain a final ranking of the algorithms, we proceed as follows. First, all algorithms that are not significantly worse compared to other algorithms are given the first rank and removed from the list. Each of the remaining algorithms that is not worse than any other of the remaining algorithms is given rank two and removed from the list. This is repeated until all algorithms have been ranked. If more than one algorithm is ranked first, we choose the algorithm with the lowest resource consumption.

3.3 Conclusions

In this chapter, we discussed several algorithms for single-objective, continuous optimization and a suitable taxonomy to distinguish different approaches by assigning algorithms to algorithm families and classes. This allows a proper specification of an algorithm portfolio capable of addressing several different problem classes, which is important for automatic algorithm selection for previously unknown problems.

We discussed parametric and non-parametric approaches to statistically analyze the performance of algorithms on randomly selected test instances from a problem class. Parametric tests, such as ANOVA, provide a powerful tool for the user to understand the performance of algorithms on the problem class. However, for an automated solution, parametric statistical tests are more complex to handle and may not be feasible if test assumptions are not met. Therefore, we discussed a procedure that relies on non-parametric tests and multiple pairwise comparisons based on rank sums to obtain a final ranking of the algorithms. Since several algorithms can achieve the first rank, the lowest computational cost determines the best performing algorithm selected for optimizing the CPPS.

The results of this chapter address and partially answer the question **RQ-2**. Combined with the data-driven test instance generation and the provided algorithm taxonomy, this allows for a fully automated benchmark experiment and selection of the best performing algorithm for a problem class derived from a CPPS.

Chapter 4

Selection Mapping Function

This chapter contains work based on the following publications [53, 146]. The notation and structure was adapted to this thesis, some parts of the text were taken verbatim.

This chapter discusses the requirements for automatic selection and deployment of the best found algorithm based on experiments using one of the data-driven test instance generation methods discussed in Section 2.2. A taxonomy of feasible algorithms is defined in Section 3.1. Section 4.1 discusses and defines the necessary notation for metadata regarding both the problem to be solved and the optimization algorithms. The implementation of the algorithm that iteratively selects and tunes an optimizer for the problem at hand is given in section 4.2. This chapter concludes with section 4.3, the procedure for tuning the optimizers.

4.1 Problem Definition and Metadata Requirements

The method used to optimize the CPPS is selected automatically for ease of use. This is made possible by using declarative goals, i.e., the user specifies the goal, not the individual process steps to achieve it. However, this declarative goal must be formulated, and goals such as *Resource Optimization* are too unspecific to be translated into an appropriate processing pipeline. Therefore, we have implemented a step-by-step procedure to assist the practitioner.

The task of an algorithm in the context of an optimization problem is to find the setting of one or more control parameters $x \in \mathbb{R}^n$ that minimizes (or maximizes) a function $y = f(x)$ subject to bounded constraints:

$$\begin{aligned} \arg\min_{x \in R^n} \; & f(x) \;\; s.t. \\ & l \leq x \leq u \end{aligned}$$

where bounds $l, r \in R^n$ can be infinite. The formulation of an optimization problem typically involves at least three steps [16]:

1. Selecting control variables x,
2. choosing the objective function, and
3. identifying constraints on x.

The first and third steps result in the definition of the parameters that control the process and adapt the CPPS. For this thesis, we focus on bound-constraints only, which must be defined by the operator. We developed a simple multi-stage goal selection for the second step, the objective function formulation, which guides machine operators. This is presented through the four-stage selection for optimization:

1. The user selects the overall goal, such as **optimization**, anomaly detection, condition monitoring, or predictive maintenance. Please note that this thesis focuses on optimization only.
2. Next, the parameter and target signals are selected. Typically these stem from a source like Open Platform Communications Unified Architecture (OPCUA), and may or may not be annotated. OPCUA is a client-server communication protocol for industrial communication and has been standardized in IEC 62541 [59]. It enables a multi-vendor data exchange across several units, e.g., machines and sensor-devices. Additionally, OPCUA allows for a semantic description of the data in form of metadata. Therefore, the necessary knowledge to identify the correct and relevant signals is use-case dependent. Depending on the use case, all, many, or just a single signal must be selected to define a parameter or the target. We assume that the operator emphasize his knowledge to increase efficiency. For example, due to this knowledge, some parameters of the process that are known to be unimportant for the target can be excluded and thus reduce the search space for the optimizers.

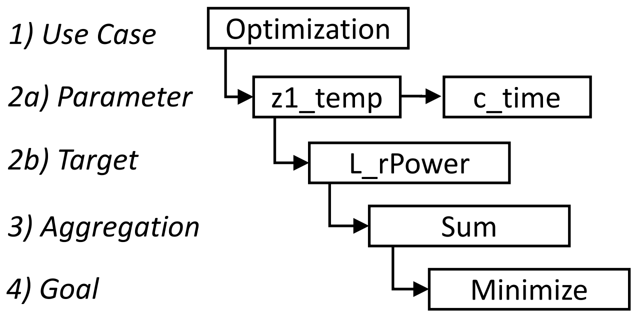

Figure 4.1: *Application of the four-stage declarative goal selection. Figure is based on [146].*

3. Aggregation functions like mean, delta, min, or max value, or the value itself, can be selected.

4. The user selects the optimization goal, e.g., minimization or maximization.

This four-step process allows the user to select the optimization goals more abstractly. This is exemplified in Figure 4.1 on an energy optimization use-case of an injection molding process. The chosen parameters (2a) are the melt temperature (z1_temp) and the cooling time (c_time), and the target signal (2b) is the power consumption (L_rPower). The goal of the use case is the minimization of the sum of the power consumption. Available algorithms are described using a standard schema. From a knowledge representation perspective, these are frames [104]. A frame represents an instance (in this case, an algorithm) using a pre-defined schema; thus, each frame slot can be interpreted semantically. Furthermore, relations between frames can be defined as the connection to a declarative goal. The schema defined for this work is shown in Table 4.1. The presented properties can be divided into mandatory properties that must have a specific value to enable algorithm usage, such as the property *use case class*. Properties in the middle section are also relevant for selection purposes, as they describe some hard constraints, e.g., the minimum number of data needed, or the possibility of being set up as running in parallel. Other properties represent the performance of the algorithms on a specific benchmark, and thus, the best matching algorithms can be chosen, such as the property *performance*. Please note that this is a relative value based on the comparison with other feasible algorithms, which may change if other algorithms are added. The required *input data type*, *output data type*, and the *use case class* are mandatory properties. They enable the system to determine which algorithms are feasible for a certain processing pipeline. The property *algorithm class* is used as a high-level property for the selection process.

Using this structure, a knowledge base to compose and configure feasible processing pipelines can be set up. A processing pipeline is a sequence of several methods implemented in modules. The preprocessing could, e.g., consist of two steps: The imputation of missing data (if any) and data aggregation to a production cycle. Dependent on the use case, this can be followed by a processor like an optimizer or a

Table 4.1: *Properties of Algorithms, italic properties can be updated by the cognition module. This thesis focuses on data types, use cases, and algorithms in bold face as they address the parameter optimization for CPPS.*

Property	Values
Input data type	**Continuous**, discrete, hybrid
Output data type	**Continuous**, discrete, hybrid
Use case class	**Optimization**, Condition monitoring, Anomaly detection, Diagnosis
Algorithm class	**One-shot**, **Monotonic Descending**, **Non-Monotonic Descending**, **Population**, **Surrogates**
Use multithreads	true, false
Min Training Data	0..n
Prefer usage	true, false
Avoid usage	true, false
Performance	0..1
Computational effort	0..1
RAM usage	0..1

model training step, e.g., for classification or regression. In this work, we use the YAML Ain't Markup Language (YAML), which is a data serialization language commonly used as an input format for various software applications. It can be used to declare resources. Alternatives to YAML include JavaScript Object Notation (JSON) and Extensible Markup Language (XML). YAML is also used to describe resources for Kubernetes and is known for its improvement in human readability compared to JSON or XML. The representation structure describing an algorithm instance is based on the 4-step process to define declarative goals, presented earlier in Fig. 4.1. Listing YAML 1 shows the exemplary knowledge representation of the Random Forest algorithm. Surrogate model-based optimization can already be defined as a pipeline consisting of at least two processors: E.g., the Random Forest model and an optimizer searching the model for promising candidate solutions. It can already be seen that several methods can be reused in other (different) pipelines. Thus, an algorithm can be listed at several locations in the knowledge base. Lines 1-3 represent the declarative user goals, and the following lines describe the algorithms suited to achieve the individual goals. Algorithms are described with the parameters, metadata about the algorithm, and the required input. Parameters (lines 6-11) are described by the type, minimum, maximum, and default value of the parameter. This also functions as the search space for tuning the algorithms, described in Sec. 4.3. The metadata (lines 12-20) represents Table 4.1 for a specific algorithm. Not initialized values are indicated by -1. Those values are determined during the run time of the algorithms. The best

YAML 1: Knowledge Representation

```
 1 Optimization:
 2   minimize:
 3     Minimum:
 4       Algorithms:
 5         Random Forest:
 6           parameter:
 7             NumberOfTrees:
 8               type: int
 9               default: 100
10               min: 1
11               max: 1000
12           metadata:
13             Class: Surrogate
14             Performance: -1
15             Computational effort: -1
16             RAM usage: -1
17             Use multithreads: false
18             Min training data: 5
19             Prefer usage: true
20             Avoid usage: false
21           input: preprocessed data
```

algorithm for the problem gets the value 1, and the worst-performing algorithm gets the value 0 as a result of a normalization of the achieved performance results. The performance is evaluated with one of the metrics discussed in Sec. 3.2. The resulting ranking can then be used to select the best algorithm for solving the configured use case. Please note that this evaluation is use-case dependent. The input data type for the algorithm (line 21) is described as a string, e.g., preprocessed data. All algorithms that compute an output that matches the given input type are candidates for a preceding module in a pipeline. Starting from the desired output for the use case, e.g., a minimization, the selection process can be repeated until the required input is designated as raw data, which marks the end of a pipeline. Thus, the representation enables a fast and easy composition of feasible pipelines and provides all necessary information to evaluate pipelines and decide which is the best suited for the production process.

4.2 Algorithmic Implementation

Depending on the available computation resources, the dimension of the data, and the amount of available data, the computational effort of each pipeline can be rated. Since resources are limited, it is required to consider whether one pipeline with good results but high resource usage or several pipelines with a lower chance of good results should be further evaluated. In an extreme case, it might be impossible to apply a certain pipeline or retrieve a new parameter suggestion during a production cycle if the computational resources are very limited. So the computational resources are significant criteria for the selection of suitable pipelines.

To rate the quality of results for a certain pipeline, experiences from previous experiments can be used. Therefore, the quality of each experiment is evaluated, and the related quality parameter in the knowledge base is adapted. When a production process is set up or adjusted, often some data will be generated or is already available, e.g., using a design of experiment. The findings from previous experiments can only be meaningfully selected from the same type of machine since it is not exactly known which characteristics influence the quality of results.

The procedure to gather or use available data, generate test instances, perform a benchmark and tuning experiment upon a list of feasible pipelines, and select the best pipeline to optimize the CPPS is presented in pseudo code in Algorithm 1. At first, needed variables are defined (lines 1-5). The parameter $\vec{x}$ is defined to store the currently applied parameter configuration of the CPPS. If no historical data are available, an initial design representing a list of different parameter configurations is created, the parameters are applied to the CPPS, and the resulting data are stored in the list d. For each entry, this list contains a parameter configuration $\vec{x}$ and the associated performance $\vec{y}$.

As several design methods exist and due to the modular design of the architecture, which is presented in Sec. 5.3, the design method can easily be adapted for special purposes of the CPPS. A Latin hypercube design is the default configuration due to its beneficial properties towards GPMs. For an overview of several design methods and corresponding optimality criteria, we refer to [5, 66, 107].

After the initialization phase, algorithm selection based on data-driven simulation, benchmarking, tuning, and selection starts within an infinite loop (line 13). Since we do not want to change pipelines in every iteration, a user-defined step size θ and a variable ζ are checked to potentially alter the pipeline only after each θ steps or in situations of prolonged stagnation or performance decrease. Then, a test function set s is generated based on the data d (line 16). Currently, the three strategies presented in Sec. 2.2.3-2.2.5 are available. The default is the conditional one-stage GPS. Simulation intends to reproduce the covariance structure of a set of samples (in this case, the process data gathered with the initial design), which maintains the topology of the problem landscape. The intention is to analyze the behavior of the performance of candidate algorithms on problems similar to the CPPS problem. The generation of several different instances allows a benchmarking of feasible algorithms. With this benchmark, the

Algorithm 1: Procedure for Algorithm Selection for Optimization

```
Input: Initial design size $s$, selection step size $\theta$, algorithm characteristics
       $KB$, goal $g$, available resources $r$, historical data $d_H$
1  define List for evaluation $e$
2  define List process data $d$
3  define List pipeline resource consumptions $p_r$
4  define Parameter $\vec{x}$
5  define $\zeta = False$
6  if ($|d_H|$ = 0) then
7      List parameter $l \leftarrow$ createInitialDesign(s, KB)
8      forall (parameter l) do
9          $d \leftarrow d$ + applyToCPPS(l)
10         $x = l$
11 else
12     $d, x \leftarrow$ historicalData $d_H$
13 repeat
14     if (nrIterations $\% \, \theta = 0 \vee \zeta = True$) then
15         $\zeta = False$
16         testFunctionSet $S \leftarrow$ generateTestFunctions(d)
17         List $p \leftarrow$ determineFeasiblePipelines($KB, g, d$)
18         $p \leftarrow$ selectCandidatePipelines($KB, r, p, e$)
19         forall $p[i]$ do in parallel
20             $e \leftarrow$ applyPipeline($p[i]$, S)
21         $KB, p_{best} \leftarrow$ ratePipelines($KB, p, e$)
22     $x_{best} \leftarrow$ getBestX($p_{best}, d, x$)
23     if ($|x - x_{best}| \geq \epsilon$) then
24         $e \leftarrow$ applyToCPPS($x_{best}$)
25         $x = x_{best}$
26     $d \leftarrow$ d + receiveNewDataFromCPPS()
27     if ( since $\theta/2$ steps stagnation $\vee$ performance decrease) then
28         $\zeta = True$
29 until true;
```

algorithms can be analyzed regarding their resource consumption (computation time and memory consumption) and performance even for a larger number of production cycles. We assume that the resource consumption depends on the current machine load, the number of function evaluations to perform, i.e., the volume of the data, and the dimensionality of the problem, but not on the problem landscape structure. Consequently, resource consumption can be analyzed quite accurately using simulations.

The algorithm description from the knowledge base is used to determine a list of all feasible pipelines in line 17. Based on the available resources, the knowledge base, and possibly existing earlier evaluations, the most suitable pipelines are selected from the list of feasible pipelines in line 18. This ensures that pipelines can be excluded which are already known not to compute a result in time or are set to avoid usage by the operator. These pipelines are applied in parallel on the test function set s, which is used for parameter tuning and benchmarking pipelines (lines 19-20). Instances are drawn randomly for each step, first tuning the algorithms with an equal budget and then benchmarking on different instances afterward.

The final decision, which pipeline is used, is computed in line 21. A normalized processing time, computed by dividing the used CPU time by the runtime of a standard algorithm, is suggested by Johnson and McGeoch [75] and Weise et al. [148]. Inspired by this idea, we chose to consider a baseline comparator, e.g., the random search, as the reference algorithm. Consequently, algorithms performing worse than the baseline will be removed from this iteration.

The best pipeline in the remaining list is chosen for application on the CPPS (line 22). If the list is empty, the current $\vec{x}$ will be returned. Should the new $\vec{x_{best}}$, suggested by the selected pipeline, differ significantly from the current $\vec{x}$, the new $\vec{x_{best}}$ is applied to the CPPS for the next iteration (lines 23-25).

The new data produced by the CPPS in terms of the parameter $\vec{x}$ and the objectives $\vec{y}$ are added to d (line 26). If there is no significant improvement after half of the step size, the ζ is set to $True$ to perform an unscheduled algorithm selection cycle in the following loop sequence. This should help at the beginning of the process to recover from poor decisions.

Process Description The main goal of this thesis is not only to implement an algorithm selection for CPPS in the context of Big Data but to define an architecture providing the necessary degree of abstraction to benefit of the algorithm selection procedure in many different use cases with different requirements.

Consequently, we will examine the selection process and its impact and emerging requirements on the architecture and its components needed to embed into a Big Data Platform (BDP). First, the architecture implementation needs a cognitive module, further denoted as *cognition* module to select different algorithms and evaluate their results as described in Algorithm 1. The general workflow of the architecture is depicted in Figure 4.2. It defines several modules involved in the processing of the data. A comfortable implementation requires modularity. This allows reusability of data processing steps, e.g. the model training and optimization modules to implement SMBO algorithms. We use SMBO as an example to describe the process. SMBO uses a surrogate model to replace the expensive objective function. In the context of CPPS, expensive can refer to production cycle time or raw material usage, for example. An initial design is used to fit a data-driven (surrogate) model, such as a GPM or Random Forest. This model is sequentially updated with new observations selected by an optimizer using an infill criterion to sample the cheap to evaluate surrogate model. The cognition module receives the necessary information from a knowledge module and

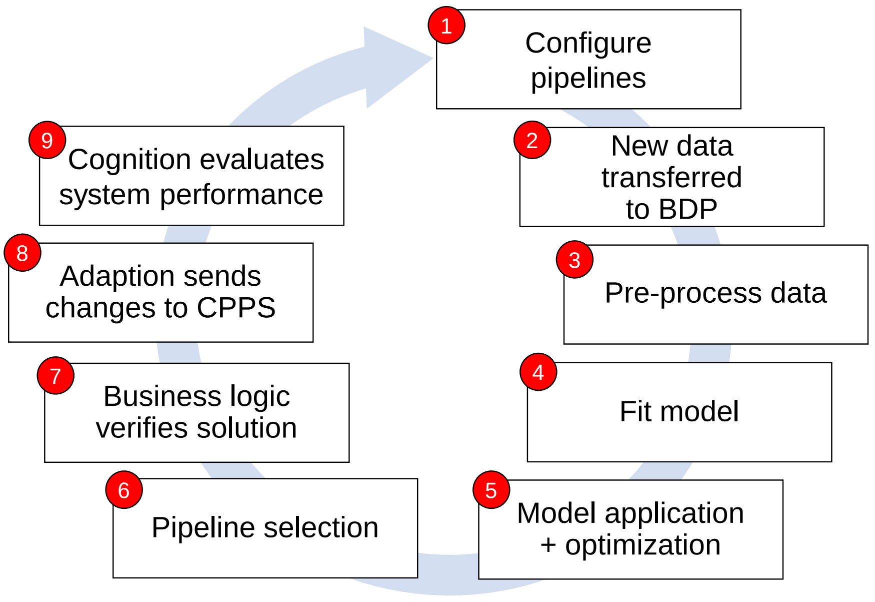
1 Configure pipelines
2 New data transferred to BDP
3 Pre-process data
4 Fit model
5 Model application + optimization
6 Pipeline selection
7 Business logic verifies solution
8 Adaption sends changes to CPPS
9 Cognition evaluates system performance

Figure 4.2: *The workflow represents nine steps that are continuously performed to adapt the pipelines and increase their performance over time.*

starts the workflow, which consists of the following nine steps:

1. The *cognition* module initializes candidate pipelines for parallel processing by varying model types and parameters. The knowledge module provides the required information about feasible algorithms and boundary constraints. Suitable models for this example use case are, e.g., GPM or random forest.

2. A *protocol translation* module transfers the data from a server, e.g., a OPCUA Server, on the CPPS to the BDP. As different types of data sources and protocols are available, an entry point to transform the data is needed.

3. The *preprocessing* module cleans the raw data. If there is missing data, imputation could address this issue. Otherwise, feature generation and extraction can be performed. For the model training, we perform a data normalization only.

4. In this step, the *GPM* and *Random Forest* models fit or update their parameters to the data and send the results to an analytics data bus for further processing.

5. The *model application + optimization* module implements the sequential step of the SMBO algorithm: it searches the previously fitted model until an optimal solution is found or the maximum number of iterations is reached. The module passes the result to the analysis bus.

6. The *cognition* module decides which pipeline to choose to optimize the CPPS by comparing the model accuracy and the predicted optimum.

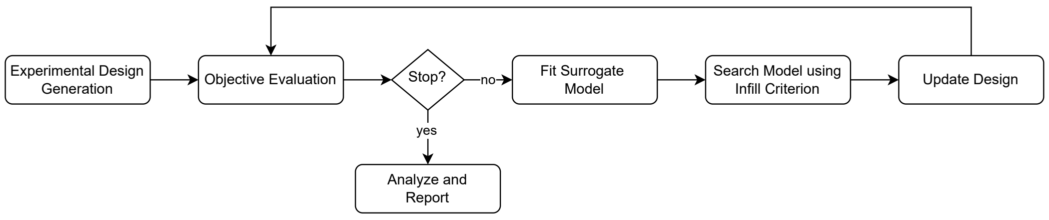

Figure 4.3: *The SPO cycle of SPOT starts on the left with design generation. Next, the design points are evaluated against the (costly) objective function and a surrogate model is fitted. Then, the model is searched for a new candidate solution using an infill criterion, such as the best predicted value or expected improvement. The design is updated with this new information and the candidate is evaluated by the objective function. The loop continues until a stopping criterion is reached, such as the maximum number of objective function evaluations or convergence.*

7. The *business logic* module checks whether the solution violates any constraints defined for the CPPS and communicates the appropriate adjustment back to the analytics bus. This is necessary to ensure the safe operation of the CPPS, e.g. in the event of a software failure involving optimization algorithms, which may be third-party software.

8. The *adaption* module translates the adaptations for the specific CPPS and sends the instructions from the BDP to the CPPS using the *protocol translation*.

9. The *cognition* module analyzes the system performance achieved with the resulting pipeline configuration from step 6. In the following steps, the impact of changes is verified and made available to the operator through information provided by the *monitoring* module.

4.3 Hyperparameter Optimization

Effective tuning of the parameters of optimization algorithms typically requires several runs of the algorithms with different configurations and is therefore considered costly (in terms of computation time). For costly objective functions, surrogate models can be an efficient solution. This involves the training of a data-driven (surrogate) model, e.g., a GPM, based upon some sample data, and sequentially updating the model with new observations selected using an infill criterion to sample the cheap to evaluate surrogate model using an optimization algorithm until a defined budget is exhausted or the process converged. Such a procedure, called SMBO, was introduced in [76]. The general workflow is shown in Fig. 4.3. An overview of several SMBO algorithms using different infill criteria can be found in [26]. In this thesis, we use the implementation of SPOT [12] for tuning the feasible optimization algorithms, due to our experience with this tool and its well known capabilities in terms of tuning. While several beneficial goals performing tuning are available [12], we aim at avoiding wrong parameter

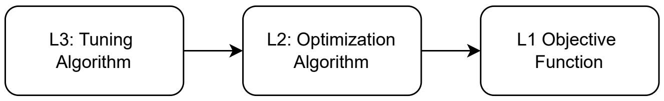

Figure 4.4: *Levels of Configurations during Tuning using SPOT. The Figure is based on [12].*

configurations and helping towards the selection of the best algorithm for real-world problems emerging from the CPPS. All feasible optimizers for a given problem will be tuned before benchmarking to select the best optimizer adapted explicitly to the problem. The tuning procedure will start when the initial exploration of the process is finished; thus, the test instances can be generated. An overview of the different levels of needed configurations occurring in tuning using SPOT is shown in Fig. 4.4. L1, as the first level, is the objective function from the real-world system, which, in this thesis, is implemented by a CPPS. The operator defines its parameters and their boundaries, which are stored in the knowledge base; see Sec. 4.1. The second level (L2) represents the optimization algorithm for solving L1. For parameter tuning, the algorithm parameter specification is needed. For evolutionary algorithms, e.g., the mutation rate, the crossover rate, the population size, and the number of offsprings are natural choices. The initial temperature and the cooldown factor are essential parameters for Simulated Annealing. Those parameters, determining the performance of the algorithms on specific problems, should be tuned. The algorithm is used to compute an optimal parameter configuration to solve the objective function in L1. The most abstract level in this scenario is the tuning algorithm (L3), which also comes with a list of parameters. Please note that there are algorithms that are not meant to be tuned, e.g., self-adaptive or internally tuning algorithms [11], which determine their optimal parameters during the search by evaluating the feedback of the objective function themselves, or by a fixed rule set, e.g., decreasing a step size parameter each n iterations. Consequently, during the algorithm implementation, the tunable parameters with their lower and upper bounds must be configured. The algorithms will be tuned on randomly drawn instances from a generated test instance set using 1-stage conditional simulation.

4.4 Conclusions

This chapter addresses the posed question **RQ-2**: How can a solution to the ASP be implemented algorithmically so that operators can optimize a CPPS online with a minimum of data science knowledge and without a hand-written procedure? We presented a step-by-step procedure for declaratively specifying the optimization problem. In addition, metadata for algorithms is also provided in a declarative way and can be combined with the specified problem to generate a list of feasible algorithms with their parameters to solve the problem, e.g., to minimize an objective such as the energy consumption of a production process. We have provided an algorithmic description of the procedure for collecting data from the CPPS, generating test instances, tuning and benchmarking algorithms to automatically select and apply the best performing optimizer based on a statistical analysis. The procedure either uses available data from the process or generates a design of experiments to be evaluated by the CPPS, and thus can be processed in a highly automated manner.

Chapter 5

Architecture for Cyber-physical Production Systems

This chapter contains work based on the following publications [23, 53, 146]. The notation and structure was adapted to this thesis, some parts of the text were taken verbatim.

A collection of requirements for an architecture for CPPS, specified based on the inspection of three different real-world use cases in the context of Cyber-physical Systems (CPSs) is given in Section 5.1. As several architectures and reference architectures exist, Section 5.2 reviews selected well-known architectures from the field of automation and cognitive science. Section 5.3 presents the developed Cognitive Architecture for Artificial Intelligence in Cyber-physical Production Systems (CAAI) and a concluding summary is given in Section 5.4.

5.1 Use Case Requirements

In the preceding chapters, we discussed several features regarding the aimed solution with a focus on continuous optimization, such as data-driven test instance generation, declarative goal configuration, algorithm portfolio specification, hyperparameter optimization, and algorithm selection based on benchmarking results. Some requirements regarding the definition of an architecture can be explicitly or implicitly derived from these previous chapters. This refers to requirements such as modularity to enable reusability, maintainability, and extensibility of algorithms in a language-agnostic way. Furthermore, a knowledge representation (for metadata about algorithms and the use cases) is needed, to enable a declarative problem configuration. Among the arguably most important benefits in using an architecture to realize a software system are flexibility, maintainability, and scalability [13]. To achieve these benefits, a broad scale of use cases addressed by the architecture are desired. Therefore, we will discuss three selected different use cases for CPS and CPPS and derive requirements for the architecture and algorithmic solutions. The first use case *Diagnosis of Modular CPPS* address a condition monitoring problem and related diagnosis in a versatile system, the second use case *Energy Efficiency Optimization in Bakeries* includes a discrete optimization problem and analysis of decentralized prediction models, and the third use case *Process Control of Concrete Spreading Machines* deals with a dynamic parameter optimization problem. The formulated requirements emerge from interviews with process engineers and experts dealing with these use cases.

5.1.1 Diagnosis of Modular CPPS

The Versatile Production System (VPS) located in the SmartFactoryOWL is a CPPS, specifically composed as a highly flexible demonstrator (see Fig. 5.1). This CPPS processes corn in different modules and finally produces popcorn. It consists of the modules delivery, storage, quality control, dosing, and production. The modules have compatible interfaces that allow different hardware configurations to enable adaptive production.

It is also possible to expand the system by adding new modules, sensors and actuators. Therefore, a self-diagnostic system is needed to detect anomalies independently of the current configuration, determine the root cause of failures, and shut down the faulty modules to prevent additional damage. A recent example of a model-based approach can be found in [24].

Due to the variety of possible combinations, no diagnostic system is implemented directly in the VPS, since today's approaches cannot efficiently handle such versatile systems. Therefore, data-driven approaches can be an efficient method for anomaly detection, as it is done in [96]. These methods have a huge potential, but are difficult to implement because they require common interface definitions and implementations have to be adapted for each specific application with some effort. A solution to implement the self-adaptation upon a model-based diagnostics must use the OPCUA stan-

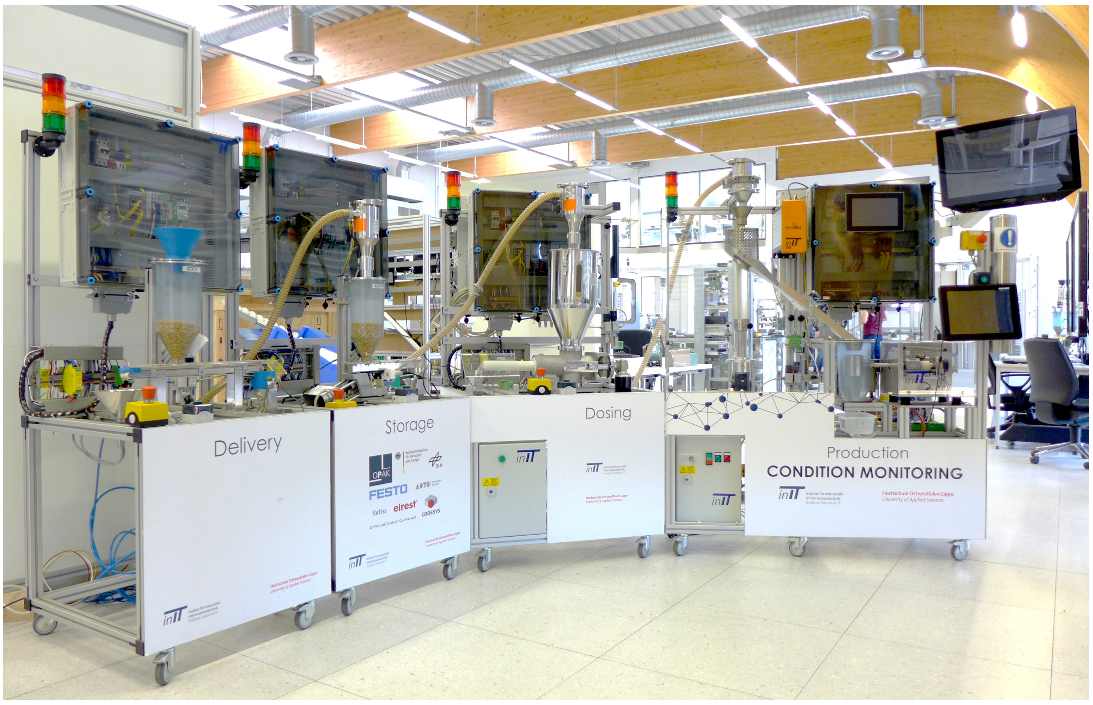

Figure 5.1: *A modular and versatile production system to produce popcorn. The modules are equipped with compatible hardware interfaces, sensors, and actuators. The system can quickly be adapted if needed.*

dard and Modbus for communication. Modbus is a communication protocol based on the client-server principle to establish information exchange between multiple hardware devices using Ethernet or serial interfaces. In terms of an architecture, the following requirements and tasks must be addressed for the VPS:

(R-A.1) Collect data via OPCUA server and Modbus.

(R-A.2) Store data to collect all lifecycle information such as process data and models of the CPPS.

(R-A.3) Perform preprocessing to handle missing values, normalize data, or adjust format.

(R-A.4) Learn diagnostic models that enable diagnosis with few training data sets for online and offline learning.

(R-A.5) Provide a diagnostic algorithm that performs a diagnosis in less than 100 ms.

(R-A.6) Decide whether a response is required and select an action.

(R-A.7) Access the controller to perform the selected action to prevent further damage.

5.1.2 Energy Efficiency Optimization in Bakeries

Optimizing the use of energy is an important task, especially in industries with high energy consumption. This scenario considers bakeries consisting of several chain stores of different types, i.e. sales only, production only, and stores combining both production and sales. Production equipment, especially ovens, are the main energy consuming equipment in these stores. Ovens contain several trays that can be controlled individually. Planning an optimal baking process with different amounts of products at different target temperatures is a crucial task for optimizing energy efficiency, as unnecessary idle and cooling times of the ovens must be avoided. A company's energy costs are not only calculated on the basis of total energy consumption, but also on the basis of the maximum power required at any given time. It is therefore advantageous for companies to spread their energy consumption over the day if possible, rather than creating large peaks by, for example, turning on all equipment at the same time at the start of the day, which is not unusual in reality. A smart startup schedule can be a significant improvement in terms of cost optimization. In addition, stores typically have large variations in the number of customers arriving at different times of the day, and thus in the required inventory of products. In addition to monitoring energy consumption, the predicted amount of products to be sold and the available inventory of products should be considered. Intelligent integration of inventory and sales systems enables the calculation of accurate decentralized models of product sales for a given time of day. Enabling the exchange of information between different models located in different stores can lead to a higher level of adaptability. The architecture must address the following requirements and tasks:

(R-B.1) Handle highly distributed and heterogeneous systems.

(R-B.2) Store data, such that appropriate historical data can be used, e.g. for optimization.

(R-B.3) Preprocess data, e.g. handle missing data and generate features such as maximum energy consumption.

(R-B.4) Learn and update models for predicting energy consumption and product demand across multiple stores.

(R-B.5) Provide simulation to enable optimization of energy efficiency and reduction of peaks in energy consumption.

(R-B.6) Provide information and guidance to store staff on energy efficient use through the implementation of an appropriate Human Machine Interface (HMI).

5.1.3 Process Control of Concrete Spreading Machines

A concrete spreading machine produces pre-cast concrete components. The mold consists of a steel pallet, casing and additional reinforcements. The machine pours concrete on the steel pallet. Casing and reinforcements mounted on the steel pallet determine the shape and properties of a component. Control parameters for the process must be set manually, which is quite difficult and requires a lot of experience, as some parameters are difficult to estimate, such as the consistency of concrete from a previous pour mixed with fresh concrete. In addition, changing parameters does not result in an immediate response because the concrete moves slowly. The goal of this use case is to learn a model that can control the process to optimize two conflicting goals: minimizing production time while maximizing the quality of the resulting product. The quality itself is influenced by many factors, such as the speed of the machine or the amount of concrete in the tank and the time since mixing. In order to implement a system that learns from process data to control and optimize production output, the architecture requirements are as follows:

(R-C.1) Acquire sensor data and perform preprocessing to extract standardized process information.

(R-C.2) Store relevant production process information.

(R-C.3) Learn models from both historical and current sensor data to control the process and account for trade-offs.

(R-C.4) Make decisions in real time or near real time to be useful in a real production process.

(R-C.5) Verify results in a simulation before allowing the algorithm to control an actual concrete spreading machine.

(R-C.6) Communicate with the machine in a standardized format for compatibility with different models/manufacturers.

5.1.4 General Requirements

The goal of our architecture is to cover versatile use cases with a focus on automatic algorithm selection for parameter optimization in CPPS. Implementations of the architecture can be easily adapted to different use cases, which greatly reduces engineering costs. In a particular use case, the adaptation could be the selection of an appropriate algorithm based on the specifications given by the user. The architecture must allow users to implement decisions and learn which decisions are promising for which application. To achieve this with a high level of automation, some additional requirements arise that cannot be directly derived from the use cases described above:

(R-D.1) Receive declarative goals from the user.

(R-D.2) Specified interfaces are well defined.

(R-D.3) Strategies for selecting an appropriate algorithm.

(R-D.4) The system learns from experience.

(R-D.5) Thorough knowledge representation.

The user should be able to easily define goals. If the system accepts declarative goals (R-D.1) such as "minimize energy consumption", the system can achieve this goal by applying and evaluating available methods with the focus on improving the result, without the need to implement an explicit sequence of commands for each goal. In addition, the user should be able to specify constraints, such as response time limits or bounds on control parameters. Furthermore, the architecture should have well-defined interfaces (R-D.2) with a thorough description of what data will be transferred to ensure modular extensibility and data consistency. Reusability and adaptability benefit from such a modular structure, and this will also enable concepts such as Software as a Service. In addition, the exchange and purchase of services between vendors is possible. The architecture requires a strategy for selecting an algorithm (R-D.3) that can efficiently produce a feasible result under the current conditions, such as the volume

Table 5.1: *Consolidation of requirements for an architecture in automation.*

Result	Source	Description
R.1	R-D.1	Receive declarative goals from the user
R.2	R-D.2	Specified interfaces are well defined
R.3	R-D.3	Strategies to select a suitable algorithm
R.4	R-D.4	The system learns from experiences
R.5	R-D.5	Thorough knowledge representation
R.6	R-A.1, R-B.1, R-C.1	Acquire data from distributed system
R.7	R-A.2, R-B.2, R-C.2	Store and manage acquired process data and models
R.8	R-A.3, R-B.3, R-C.1	Perform preprocessing
R.9	R-A.4, R-B.4, R-C.3	Learn a model from data, might be time and resource-limited
R.10	R-A.5, R-B.5, R-C.5	Perform a model analysis which might have a limited response time
R.11	R-B.6	Interaction with the user (HMI)
R.12	R-A.6, R-C.4	Decision making, e.g. send new parameters to the controller
R.13	R-A.7, R-C.6	Apply the action on the controller

of data, the required response time, or the nature of the problem to be solved. A cognitive architecture reflects on decision making and is able to learn from past experience (R-D.4) to improve processing efficiency and outcome over time. To support this decision making and to model/store additional learned knowledge, it is necessary to implement an appropriate knowledge representation (R-D.5) that represents information about machines and processes, newly learned rules, and expert domain knowledge.

Please note that this list of requirements is not meant to be complete or exhaustive, but it gives an overview of some important concepts needed to address several use cases in the context of CPS. In the following section, we review existing known architectures with a focus on the requirements discussed above.

5.2 Evaluation of Existing Architectures

There are at least two different classes of architectures related to this work, namely reference architectures from the field of automation and cognitive architectures from the field of cognitive sciences. We do not consider models or architectures that are used to classify the current level of digitalization or automation of a company or parts of a company, such as manufacturing, which can be done by applying, for example, the *Reifegradmodell Industrie 4.0* [74] or the *Fraunhofer Industrie 4.0 Layer Model* [118]. The reference architectures in automation represent a generic structure for a class of architectures that should help to design automation systems. We chose the well-known Reference Architecture Model Industrie 4.0 (RAMI4.0) [17, 1], Industrial Internet Reference Architecture (IIRA) [95], and 5C architecture [90].

In comparison, cognitive architectures steming from the field of cognitive sciences have been developed to understand and reproduce human cognition. Such a cognitive architecture is defined in [92] as *'a theory of the fixed mechanism and structures that underlie human cognition.'* According to Neisser [116, p.4], cognition refers to "all processes by which the sensory input is transformed, reduced, elaborated, stored, recovered, and used". Regarding the context of Industry 4.0 (I4.0), we define cognition as follows.

Definition 4 (Cognition)**.** *Cognition refers to all the processes by which the input data is transformed, reduced, elaborated, stored, recovered, and used to make decisions in terms of selecting methods and monitoring results for improvement in solving I4.0 use cases such as condition monitoring, anomaly detection, optimization, and predictive maintenance.*

Note that this is a rather abstract definition and cannot be considered equivalent to human cognition, as a system cannot yet generate or invent completely new methods to solve a use case. Our goal is to enable cognition with respect to the parameter optimization problems in CPPS by providing methods for selecting from available tools and learning from applications for future improvements.

5.2.1 Reference Architecture Model Industrie 4.0 (RAMI4.0)

In 2013, the three German associations Bitkom, VDMA, and ZVEI founded the *Plattform Industrie 4.0* to merge the different interests and requirements of electrical engineering, mechanical engineering, and information technology into a common model for Industry 4.0. The resulting reference architecture model, shown in Fig. 5.2, was published in 2015 [17, 1]. The model consists of three dimensions to represent the most important aspects:

- Layers: The six layers are based on the Smart Grid Architecture Model [28] and have been adapted to meet the requirements of I4.0. The layers represent the digital representation of an asset, such as a machine. The digital representation includes the specification of the asset, but also a description of its functional and communication behavior. The complexity of the layers increases up to the *Functional* and *Business* layers. These provide a runtime and orchestrate the services that support the business processes. The layers are loosely coupled and events and data are intended to be exchanged within a layer or between adjacent layers[1].

- Life Cycle & Value Stream: The horizontal axis represents the life cycle of assets and products in the I4.0 environment based on IEC 62890 and distinguishes between types and instances.[1].

- Hierarchy Levels: This dimension classifies assets in terms of their I4.0 functionality and responsibilities within a production facility. The classification is

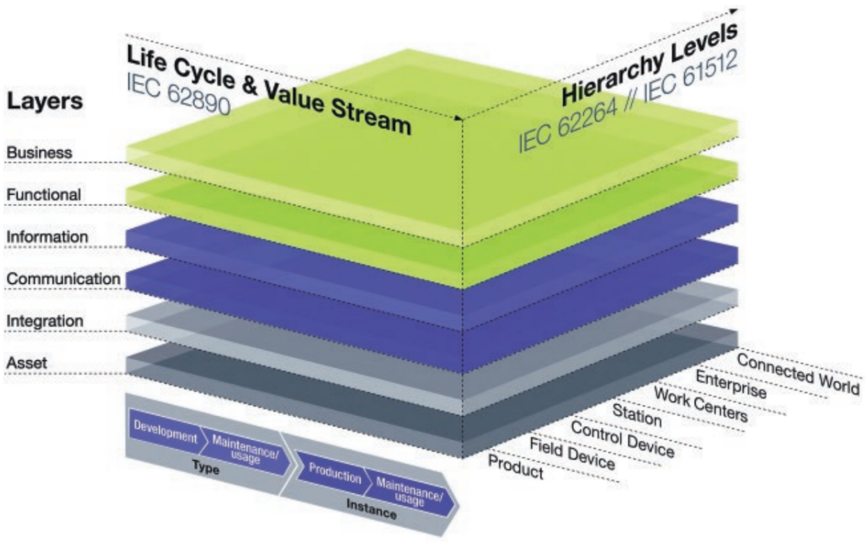

Figure 5.2: *Reference Architecture Model Industrie 4.0 (RAMI4.0). The Figure is based on [1].*

based on IEC 62264 and IEC 61512 and extends the automation pyramid by introducing the additional levels *product* and *connected world* to meet the new requirements [17].

Consequently, RAMI4.0 can be used to classify machines and components of a CPPS and provides a standardized approach for the design of I4.0 machines. This intention was confirmed when the first related model was presented in the same publication. The "Industrie 4.0 Component" contains communication functionality and an administrative shell [1, 122]. Such an I4.0 component could be an entire production system, a single machine, or even just a module of a machine. Existing assets can be upgraded to I4.0 components because the additional functionality can be provided by an external system.

5.2.2 Industrial Internet Reference Architecture (IIRA)

The Industry IoT Consortium (IIC) introduced its IIRA in 2015 and updated it to version 1.10 in 2022 as a standards-based architectural template and methodology to enable Industrial Internet of Things (IIoT) system architects to design their systems based on a common framework and concepts [95]. Figure 5.3 shows its three dimensions, comparable to the RAMI4.0.

- Viewpoints: As a result of the analysis of various IIoT use cases, e.g. developed by the IIC, relevant stakeholders and their concerns are processed and mapped to four viewpoints (business, usage, functional, and implementation).

- Lifecycle Process: The IIRA is a system conceptualization tool that highlights important system concerns that may affect the lifecycle process. Through its viewpoints, it provides guidance for specifying a system lifecycle process from

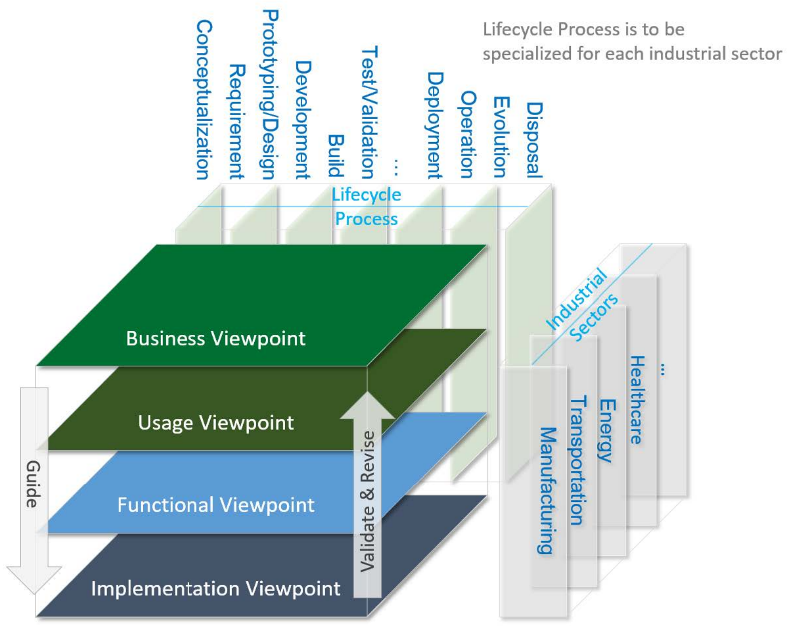

Figure 5.3: *Scheam of the Industrial Internet Reference Architecture. The Figure is based on [95].*

IIoT system conception through design and implementation to system disposal, specialized for a specific use case.

- Industrial Sectors: The related stakeholders in each viewpoint need to consider the affected industrial sectors to capture and describe the specific concerns that result in the unique system requirements.

The IIRA summarizes common concerns from different stakeholder perspectives gathered from many use cases and projects, and therefore represents a high level of abstraction. A central idea of the IIRA is the need to network larger systems and to establish control over hierarchies of machines. Therefore, this architecture seems well suited for Industrial Control Systems (ICS). Accordingly, typical IIoT systems are decomposed into five functional and interconnected domains (control, operations, information, application, business). The IIRA is intended as an iterative top-down process model to describe and develop architectural concerns on each viewpoint by its stakeholders. The results of one viewpoint serve as a guide and impose requirements on the viewpoint below, while discussion of the concerns in a subsequent viewpoint may validate or cause a revision of the decisions in the viewpoint(s) above. However, the IIRA remains at a high level of abstraction to support broad industry applicability. System architects can use and extend the architectural results of the implementation viewpoint as the basis for a concrete system architecture. For this purpose, the IIRA describes some suitable architecture patterns, mainly based on the three-tier architecture pattern, which must then be adapted, extended and specified by architects for a specific use case.

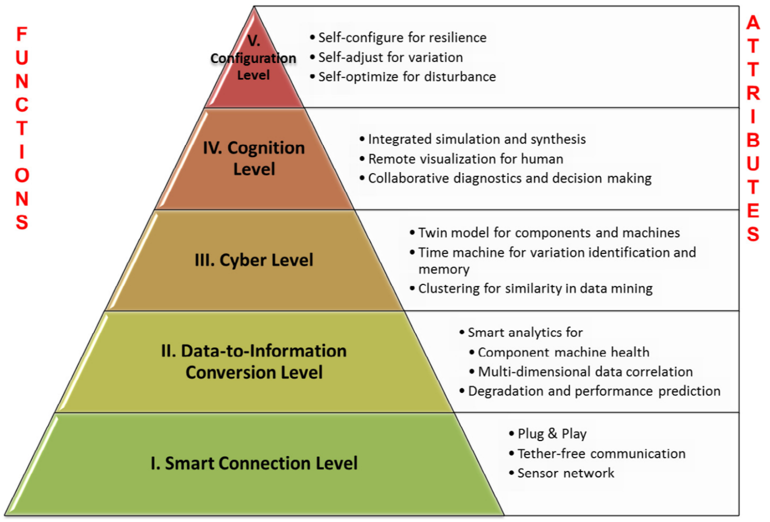
FUNCTIONS
ATTRIBUTES
V. Configuration Level
• Self-configure for resilience
• Self-adjust for variation
• Self-optimize for disturbance
IV. Cognition Level
• Integrated simulation and synthesis
• Remote visualization for human
• Collaborative diagnostics and decision making
III. Cyber Level
• Twin model for components and machines
• Time machine for variation identification and memory
• Clustering for similarity in data mining
II. Data-to-Information Conversion Level
• Smart analytics for
• Component machine health
• Multi-dimensional data correlation
• Degradation and performance prediction
I. Smart Connection Level
• Plug & Play
• Tether-free communication
• Sensor network

Figure 5.4: *5C architecture for implementation of cyber-physical systems. The Figure is based on [90].*

5.2.3 5C-Architecture

The 5C architecture is introduced by Lee et al. in [90] and focuses on I4.0-based manufacturing systems. 5C stands for the five levels: connection, conversion, cyber, cognition, and configuration. The architecture can be seen as a guideline on how to reach the goal of cognition (called self-x in the context of 5C), starting with initial data acquisition. Self-x refers to a set of techniques needed for efficient operation, such as self-configuration, self-diagnosis, or self-optimization, where the *self* means that the task is performed automatically by the CPS. Consequently, this architecture ultimately aims to provide cognitive functionality to autonomously operate the CPS. The architecture is presented in Fig. 5.4. At the *Smart Connection Level*, data from multiple sources such as sensors, controllers, Manufacturing Execution Systems (MES), or Enterprise Resource Planning (ERP) is collected by a central server. In addition, characteristics such as sensor signals are selected at this level. The *Data-to-Information Conversion Level* uses algorithms to process the data and generate information, such as health values or remaining useful life. Typically, machine learning techniques are implemented at this level. The information from many machines is collected in the *Cyber Level*, which acts as a central information hub. It can be used to compare machines, make predictions, or get more insight from a machine by combining information or using historical information. The *Cognition Level* provides deeper insights, especially to the user, who gets comparative information as well as information about individual machines. In the first publication [90], it is described as a resilient control system that gives feedback from the cyber space to the physical space, while in later publications, such as [91], decision making is the focus of this layer. It can be used, for example, to optimize maintenance by prioritizing tasks or logistical planning [91]. *Configuration Level* is the top layer that provides feedback from cyberspace to physical space and is a

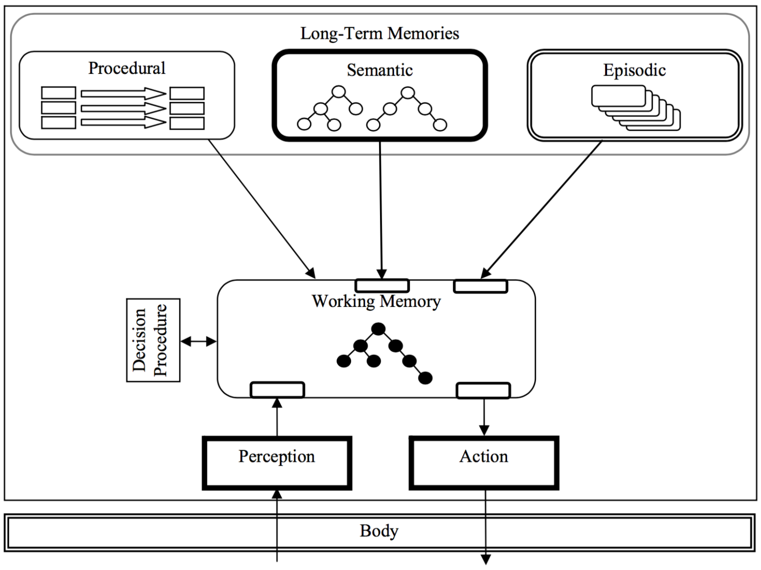

Figure 5.5: *Simplified structure with focus on memories of Soar 9. The figure is based on [119].*

supervisory control necessary for self-configuration and self-adaptation. In particular, it can be used for optimization or product quality improvement.

5.2.4 Soar and ACT-R

The development of Soar was motivated by the study of the requirements for general intelligence based on the theory of human cognition. The first version of Soar was created in 1982 by Laird et al. [87], but it is still under development and an active area of research [86]. This brief review focuses on the most basic components and features. Memory in the Soar architecture, shown in Fig. 5.5, is divided into long-term memory and working memory. The working memory consists of objects representing states, while long-term memory consists of procedural memory, semantic memory, and episodic memory. To perform a task, there is an initial state, a target state, a problem space, and operators that can be applied to change the states. Production rules from procedural memory are used to suggest an operator with certain preferences that can be applied. A particular operator is applied to the state. This sequence is repeated until the target state is reached. If no suitable operator could be identified (called an impasse), a subgoal is created and solved by trial and error. Impasses are caused by lack of knowledge. Once an impasse is solved, chunking is used to preserve the knowledge by creating new production rules to avoid running into the same impasse again.

In Adaptive Control of Thought-Rational (ACT-R) theory [3], cognition is considered as the result of an interaction between procedural and declarative knowledge. Procedural knowledge is represented in units called production rules, which consist mainly of a goal, actions, and conditions. Declarative knowledge is represented in the form of chunks. The high degree of connection between the two becomes clear as

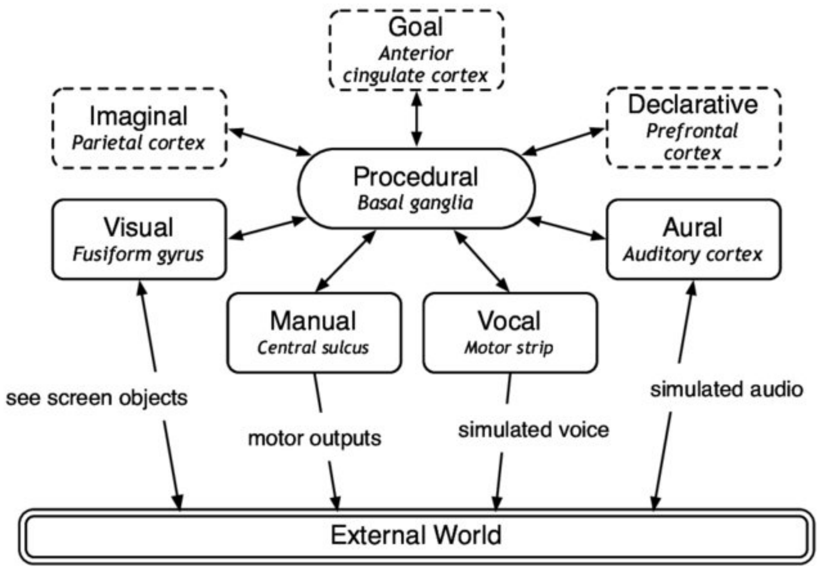

Figure 5.6: *ACT-Rs architecture, distinguishing declarative and procedural memory. The Figure is based on [61].*

both the state and the actions of production rules are stored as declarative knowledge. Anderson's basic idea behind this theory is that, in order to achieve cognition and intelligence, "the whole is no more than the sum of its parts, but it has a lot of parts" [3, p.356]. Anderson states that intelligence is the result of the collection and coordination of many small units of knowledge. The main questions that are the focus of current ACT-R research are the following:

- How are these units of knowledge represented?
- How are they acquired?
- How are they used in cognition?

An overview of the ACT-R architecture is given in Fig. 5.6. The visual and aural interfaces to the external world are responsible for creating declarative knowledge chunks by appropriately encoding the environment. The manual and vocal functions connected to the external world provide the ability to perform actions on the environment, such as steering movements in a car driving situation or using a computer keyboard to enter a solution to a math equation. The potentially large amount of knowledge accumulated during the application of cognition is only partially activated during a task, depending on its context and prior success. This allows efficient access even when the amount of data collected is quite large.

5.2.5 Evaluation Results

An overview of the evaluation of the examined architectures with respect to the requirements is given in Table 5.2. The requirements were previously summarized Table 5.1.

Table 5.2: *Evaluation results for the architectures. '-' indicates that the requirement is not addressed by the architecture, 'O' means that it is not sufficiently addressed, e.g. there may be some additional effort needed, and '✓' indicates that the architecture fulfills the requirement. The requirements are listed in Table 5.1.*

Requirement	RAMI4.0	IIRA	5C	SOAR	ACT-R
R.1	-	-	-	✓	✓
R.2	O	O	-	-	-
R.3	-	-	-	O	O
R.4	-	-	-	✓	✓
R.5	✓	-	-	✓	✓
R.6	✓	✓	✓	-	-
R.7	✓	✓	O	-	-
R.8	✓	✓	O	-	-
R.9	O	✓	✓	-	-
R.10	O	✓	✓	-	-
R.11	✓	O	-	-	-
R.12	O	✓	✓	✓	✓
R.13	-	✓	✓	✓	✓

For a more detailed discussion of this evaluation, we refer to [23]. As can be seen, none of the architectures presented fulfills all of our requirements for a cognitive architecture for CPPS. In Fig. 5.7 we have graphically classified the architectures according to their level of abstraction and their generalizability. A high level of abstraction means that there is a lot of freedom in the implementation (which we consider as a higher level of effort at the same time), while a high level of generalizability means that the architecture can be easily adapted to many different use cases. The cognitive architectures have a low level of abstraction because there are fixed structures for implementing use cases. Although the architectures are cognitive, they need to be adapted for different use cases because each use case requires different knowledge. Especially requirements towards big data, e.g., horizontal and vertical scalability or expandability, are (of course) not addressed by the cognitive architectures. In comparison, the reviewed automation architectures are considered more abstract. There are no strong constraints or specifications on their implementation, which adds more challenges and effort to an implementation. However, they can be applied to many use cases. The 5C architecture, with its focus on manufacturing and self-optimization, is less generic and less abstract than the other automation architectures, but we still consider it too abstract to be easily used for the use cases described. It is important to rely on well-defined interfaces. This allows for modularity and thus reusability of existing code, and maintains data consistency. This need is also mentioned for RAMI4.0 [1] and for IIRA [95], where interfaces are defined in the implementation view. We identified a gap: An architecture with a low level of abstraction to reduce implementation effort, and a high level of generalizability to solve many CPPS use cases. We conclude that none of the

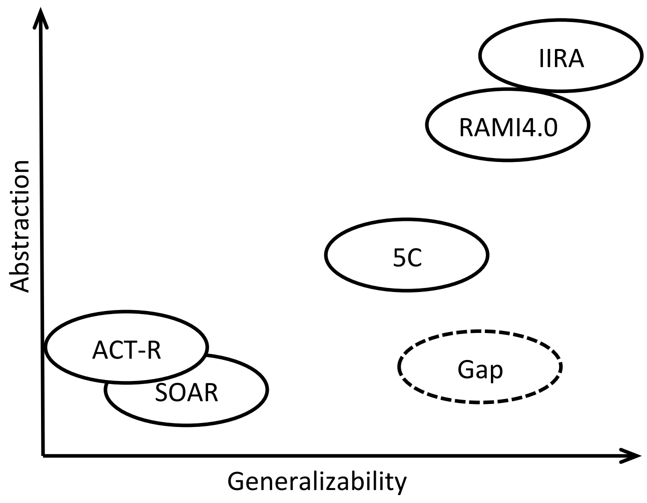

Figure 5.7: *Cognitive architecture evaluation result*
Visualization of the gap in current architectures

evaluated architectures are directly suitable for an implementation to solve, e.g., a parameter optimization problem in CPPS. Addressing the gap can probably be achieved by combining the features of the two types of architectures evaluated, a big data architecture for CPPS with cognitive capabilities in terms of automated algorithm selection for declaratively configured goals. In the next section, we address the identified gap and present concepts, methods, and technologies that comprise a new architecture to solve CPPS parameter optimization use cases in a highly automated manner.

5.3 CAAI—A New Cognitive Architecture for Artificial Intelligence in Cyber-physical Production Systems

This section presents our cognitive big data architecture for CPPS. First, we summarize the goals and methods of our architecture and present solutions and technologies to implement these methods.

5.3.1 Goals and Methods

Our goals (G) for the application of CAAI are represented by several requirements that emerge from different use cases and companies and that need to be fulfilled by an Artificial Intelligence (AI) solution.

(G-1) **Reliability:** In a competitive market environment, the efficiency of the CPPS is important, and reliability is a prerequisite for achieving it. Since the CAAI supports the CPPS, the two are interconnected and share the same requirements.

Capturing the complete data is essential, as it contains a great deal of value, especially when it contains information about the quality of the products produced, which can be used by AI applications. To avoid data loss or downtime of the CPPS, the CAAI and its underlying infrastructure must be stable and reliable [135, 45].

(G-2) **Flexibility:** A major drawback of existing AI solutions is that they are often developed by an AI expert for a single machine or a single problem. As a result, these solutions do not include common interfaces that allow for adaptation or extension of the existing system. This inflexibility is unacceptable, because there is a great demand in the market for quickly adaptable CPPS, which cannot be satisfied by the approach of specific AI solutions. The CAAI has to be flexible and extensible in order to react quickly to this market demand [25] and for future developments, e.g. in the field of big data algorithms.

(G-3) **Generalizability:** The CAAI should apply to many types of CPPS and support many different use cases. Since it is not possible to pre-select algorithms for all combinations of system types and applications, their selection should be automatic in a declarative manner. Thus, the CAAI must process the user-defined goals, derive a feasible process pipeline for the specific system, and learn over time to improve the system's performance, i.e., the CAAI implements *cognitive* capabilities [123].

(G-4) **Adaptability:** The realization of adaptability through the CAAI increases the efficiency of the CPPS by directly adjusting process parameters, so that users do not have to change them manually. In addition, adaptability allows for automation of the adjustment, which is less prone to error and ultimately allows for an autonomous system. However, the CAAI must ensure the safe operation of the CPPS throughout the process. For example, during optimization, the operating limits of the CPPS must be respected, while during anomaly detection, there may be machine parts that must continue operating even in an emergency. Therefore, the boundary conditions must be included in the CAAI, and CPPS adjustments are only allowed within these boundaries [123, 93].

To tackle the goals (G-1) - (G-4), the following methods (M) are considered in this approach:

(M-1) **Big Data Platform:** The continuous and reliable operation (G-1) is ensured by a BDP. The BDP includes various techniques to achieve the goal, such as orchestration, virtualization, and containerization. Orchestration instantiates and connects the selected modules, creating the necessary data processing pipelines. Orchestration also allows applications and their containers to be moved to the remaining functional infrastructure when parts of the system fail. Containerization, which provides virtualization at the operating system level, allows multiple isolated user space instances to exist. It improves reliability because each instance can only access the contents of its container and the devices associated with that container [115]. Orchestration of virtualized containers also supports horizontal and vertical scalability, since it is possible to create multiple instances of a container to work in parallel, or to assign computationally-intensive modules to computation units with sufficient resources.

(M-2) **Modularization:** A modularization of the AI components enables the flexibility (G-2) of the CAAI, e.g. to adapt the processing pipelines to specific requirements. In addition, modular components require well-specified interfaces with detailed definitions. Furthermore, the modular design reduces development and maintenance costs by integrating existing components. Only new components need to be developed.

(M-3) **Cognition:** Automated process pipeline generation methods allow the realization of different use cases without the involvement of an AI expert and the extension to different types of CPPS. Automated pipeline generation is an important feature to ensure generalizability (G-3) for specific use cases, dynamic systems and changing environments. It is realized by the cognitive module, which generates and tunes the feasible pipelines and selects the most appropriate algorithms according to the given data and the defined goal. For the goal of continuous parameter optimization, we have implemented a tuning and selection procedure based on test instance generation (see Sec. 4.2). The cognitive module evaluates the different pipelines to gather information about the performance of different algorithms for a given use case.

(M-4) **Automatic Decision:** Knowledge is required to interpret the results of the algorithms and to derive suggestions for implementing automatic or CAAI-assisted adjustments. Therefore, new parameter sets or other system changes identified by the algorithms must be applied to the CPPS. (G-4). In addition, boundary conditions may be defined and applied to the decision, such as a minimum expected improvement or specifications that ensure a safe adaption of the CPPS. Moreover, the decision has to be applied to the CPPS controller that adapts the physical machine. For the realization, several existing approaches, such as skill-based engineering, can be used [25, 160, 97].

5.3.2 Architecture Overview

A graphical overview of our architecture is given in Fig. 5.8. The CAAI-Big Data Platform (CBDP) wraps the architecture and arranges software modules in two processing layers, the Data Processing Layer (DPL) and the Conceptual Layer (CL), and connects them via three bus systems (Data, Analytics, and Knowledge Bus). This conceptual structure is based on a three-tier architecture with a focus on the data processing perspective to simplify interoperability and ensure horizontal scalability.

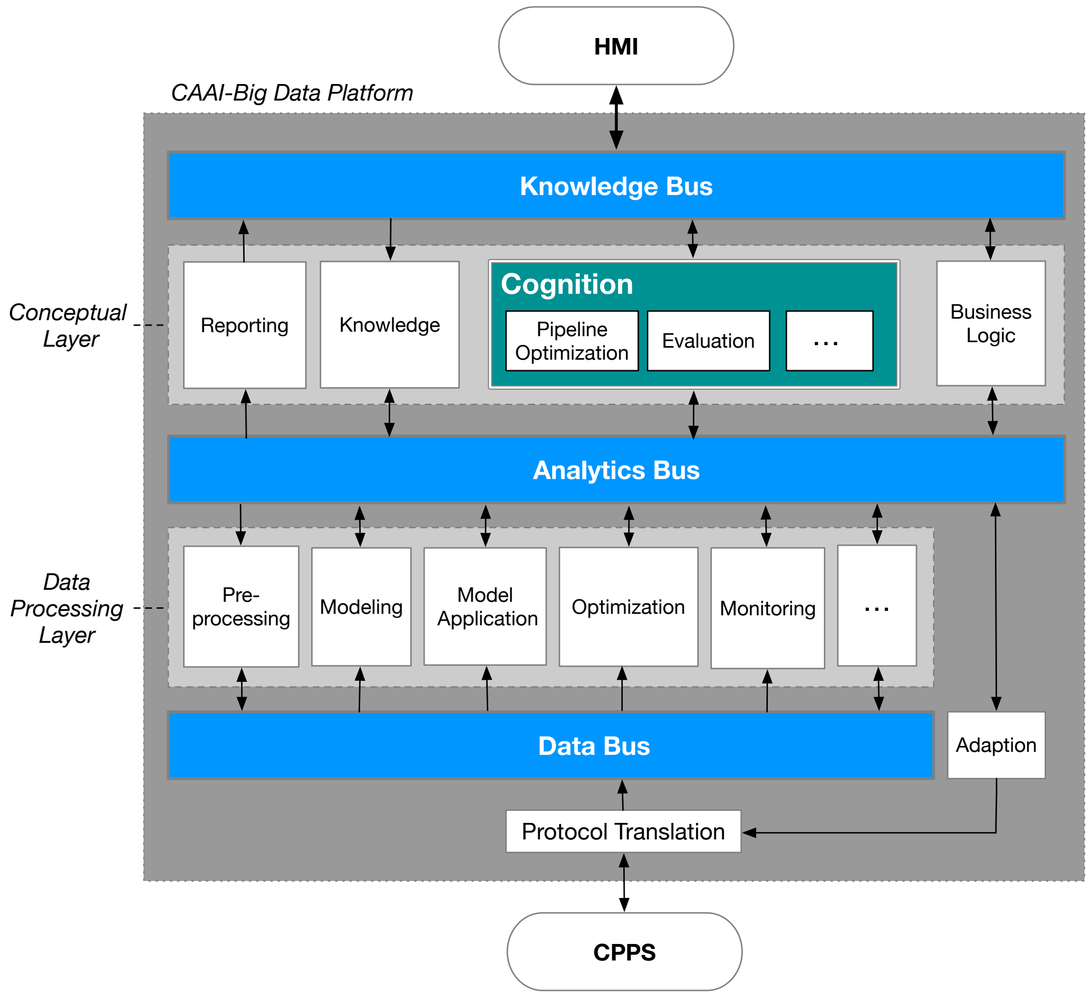

Figure 5.8: *CAAI Architecture Overview. The CAAI Big Data Platform (CBDP), shown in dark gray, manages the bus systems and layers. Bus systems are colored in* blue *and establish communication between the modules. The arrows show the intended information flow. The layers are shaded in light gray and contain the different modules. The cognition module, which provides automatic configuration, is colored in* turquoise. *Oval shapes represent external systems, such as a CPPS or a human-machine interface. The figure is taken from [53, p.614].*

The CBDP is a distributed system and can be hosted on a single machine, an on-premise cluster, or by a cloud provider. Our stated architectural goals (G1-G4) are supported by the CBDP through the implementation technologies, which are presented below. Transferring incoming/outgoing data, orchestrating the data processing tasks, assuring the persistence of results, and managing the communication between modules are the resulting tasks of the CBDP. The following concepts and technologies (T) are used to accomplish these tasks.

(T1) **Container Virtualization:** All components of the system exist as virtualized containers on the CBDP. The isolation of module requirements from the general environment ensures that all requirements for a specific module are met and do not interfere with other modules on the same platform, similar to virtual machines. Unlike virtual machines, a container uses the host operating system, and containers share binaries and libraries where possible, resulting in less overhead. Containers are consistent and immutable, ensuring compatibility across systems. A central container registry stores container images and tracks changes through versioning. Docker is used as the container engine for the implementation [115]. Images consist of all the necessary code instructions to install the requirements and create a specific environment to run the desired algorithm or software. In general, a validated, executable image guarantees that it will work the same on any computer, server, or cloud environment.

(T2) **Orchestration:** The CBDP manages the necessary infrastructure and orchestrates virtualized components to compose a system of microservices that perform a specific task. Orchestration frameworks handle the deployment, configuration, updating, and removal of the virtualized software components. A text file declaratively composes a system and lists the various services. The orchestration is done by Kubernetes, which can use the Docker container engine [65]. The *cognition* module uses the *orchestration* to instantiate pipelines with selected algorithms and evaluate the results.

(T3) **Microservices:** All modules are developed as microservices to compose the software system for a specific use case from smaller, self-sufficient parts. Each module includes standardized communication functionality to publish and subscribe to relevant topics on the bus system [147]. The resulting system is modular, language agnostic, and uses well-defined interfaces. Following microservice best practices, each microservice can store internal data in its local storage.

(T4) **Messaging:** The various bus systems managed by the CBDP transfer data via messaging. Messaging allows asynchronous communication between modules and allows for parallelization as well as multiple processing of data for different purposes via topics and consumer groups. Adding more instances to the same consumer group would result in distributed processing of incoming messages, which is useful when a task is very time consuming or response time is limited. We chose Kafka [114] as a reliable messaging system for our platform.

(T5) **Schema Management:** A schema stores the metadata of the data, with all the available fields and datatypes [31]. When a module publishes to the bus system, the serializer applies the schema and encodes the message or filters out non-conforming messages. A consumer subscribing to a topic on the bus has access to the same schema and can verify integrity before encoding the incoming message. This ensures unambiguous communication over the bus system and allows easy integration of additional modules. A central schema registry distributes and versions the schemas, allowing controlled data evolution.

The combination of technologies (T1-T5) supports the overall goals and the methods

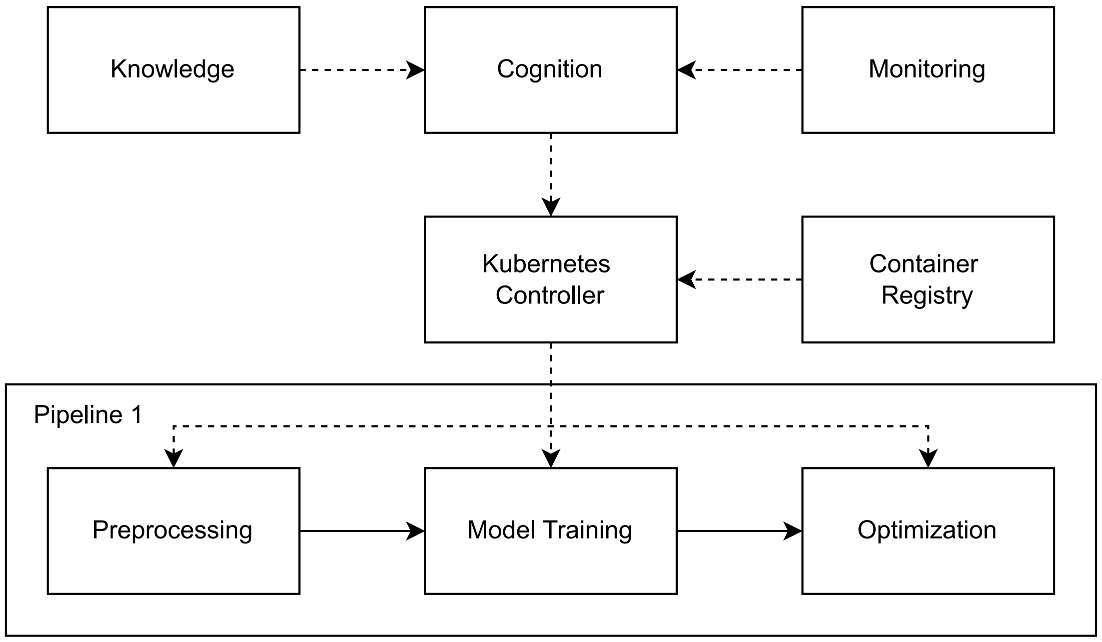

Figure 5.9: *The cognition creates new data processing pipelines based on information about the available algorithms from the knowledge module and information on current resource usage provided by the monitoring module. The Cognition decides on one or more pipelines and instructs the Kubernetes Controller to instantiate the data processing modules for each. The Kubernetes Controller loads the container images from the registry and starts all the jobs that form the data processing pipeline. The figure is based on [146, p.3524].*

to achieve them, namely to provide a reliable infrastructure (M-1) for modular development (M-2), e.g. to reuse existing modules, or to extend the system with additional algorithms. This allows the cognition module to run and evaluate additional experiments by automatically creating processing pipelines (M-3) and automatically adapting the CPPS (M-4) when a feasible and beneficial solution is found. The complete process to dynamically create a data processing pipeline is depicted in Figure 5.9.

5.3.3 Bus Infrastructure

Data bus is a term used in computer architecture and electronics to refer to a subsystem that facilitates the transfer of data between various components of a computer or electronic system. The concept of a data bus is essential to enable communication between different components in a computer or electronic system, such as the central processing unit, memory, input/output devices, and other peripherals.

In the context of this thesis, a data bus acts as a central hub for data integration, allowing data from different sources to be collected, transformed, and distributed to other systems or data consumers. An implementation must be designed to scale horizontally to accommodate increasing data volumes and support high-throughput data transfer. A data bus decouples data producers from data consumers, which means that changes in one system do not necessarily affect other systems connected to the data bus.

To implement the bus systems of the CBDP, we chose Kafka, a distributed streaming platform that is widely used for building real-time data pipelines and event-driven applications. Kafka uses topics to organize and categorize data streams, and each topic represents a specific stream of events or messages [114]. Accordingly, the data processing modules of the CBDP communicate asynchronously over three buses, the data bus, the analytics bus, and the knowledge bus. The level of data processing increases incrementally from bottom to top, and in some cases horizontally. Each bus contains multiple topics that can be subscribed to by modules connected to the bus. Kafka topics can be configured to be automatically created when producers or consumers first write to or read from them. Topics in Kafka are divided into partitions where each partition is an ordered and immutable log of messages. Partitions are the unit of parallelism and distribution in Kafka, allowing concurrent reads and writes. Additionally, the user can configure the number of replications for each topic. Replicas are copies of the same partition distributed across different nodes in the Kafka cluster. If a partition becomes unavailable, replications ensure that the data remains accessible.

All modules publish their data to the respective attached bus on a predefined topic, so that one or more other modules can use the data and intermediate results for their processing. The architecture thus implements a message-driven processing approach, resulting in a flexible and agile system with clear interfaces and hierarchies. The main features of applications that use message queuing techniques are [68]:

(i) No direct connections between modules.

(ii) Communication between modules can be time-independent.

(iii) Work can be done by small, self-contained modules.

(iv) Communication can be controlled by events.

(v) Data integrity through validation schemes.

(vi) Recovery support.

The following paragraphs provide a detailed description of the three bus systems, while Tab. 5.3 gives an overview and summarizes the estimated differences of each data type with respect to volume and velocity, computational effort, entropy, and interpretability.

Data Bus The data bus transports raw data from a CPPS, as well as data from demonstrators, external simulators, or simulation modules. Cleaned and further pre-processed data is also published back to the data bus by the pre-processing modules. Therefore, the data volume and velocity is high, although the entropy is still quite low and the interpretability is complicated. Overall, the data bus transports streams of real-time data.

Table 5.3: *Comparison of the data and estimated differences on the three bus systems used in the CAAI.*

	Data bus	Analytics bus	Knowledge bus
Type of data	Raw, Preprocessed	Processed	Enriched
Volume and Velocity	High	Moderate	Low
Computational effort	Low	High	Moderate
Entropy	Low	Moderate	High
Interpretability	Complicated	Moderate	Easy

Analytics Bus Data transported on the analytics bus have a higher information density than data on the data bus. The number of processing pipelines and the type of algorithms used determine the computational complexity. These get instantiated by the cognition module and are expected to consume a significant amount of the available processing power. Consequently, the analytics bus hosts knowledge, models, and results from the model application or optimization, the monitoring module, and the business logic to derive commands to adjust the system.

Knowledge Bus The knowledge bus enables communication between the end user and the system and combines the knowledge, business logic, and user-defined goals and actions. The cognitive module receives declarative goals defined by the user. Information from the analytics bus is condensed into reports for the user. In addition, user feedback can be requested. In this way, the knowledge bus uses enriched data, which supports interpretability and provides the most value to the user.

5.3.4 Layer

Multiple modules process the data within two layers. Each layer can be expanded individually. Each of the modules processes the data in a specific way, e.g. with a specific algorithm. Several modules are combined to enable complex data processing.

Data Processing Layer The Data Processing Layer (DPL) handles sub-symbolic data and therefore contains all modules that process sub-symbolic data. Instances of modules are combined into processing pipelines to solve a desired task based on the raw data (see Figure 5.10). Except for the monitoring module, all initially provided modules belong to one of the following four types:

(i) Preprocessing modules receive data from the *data bus* and return results to the *data bus*. They prepare data for use, for example, by imputing missing values or synchronizing timestamps.

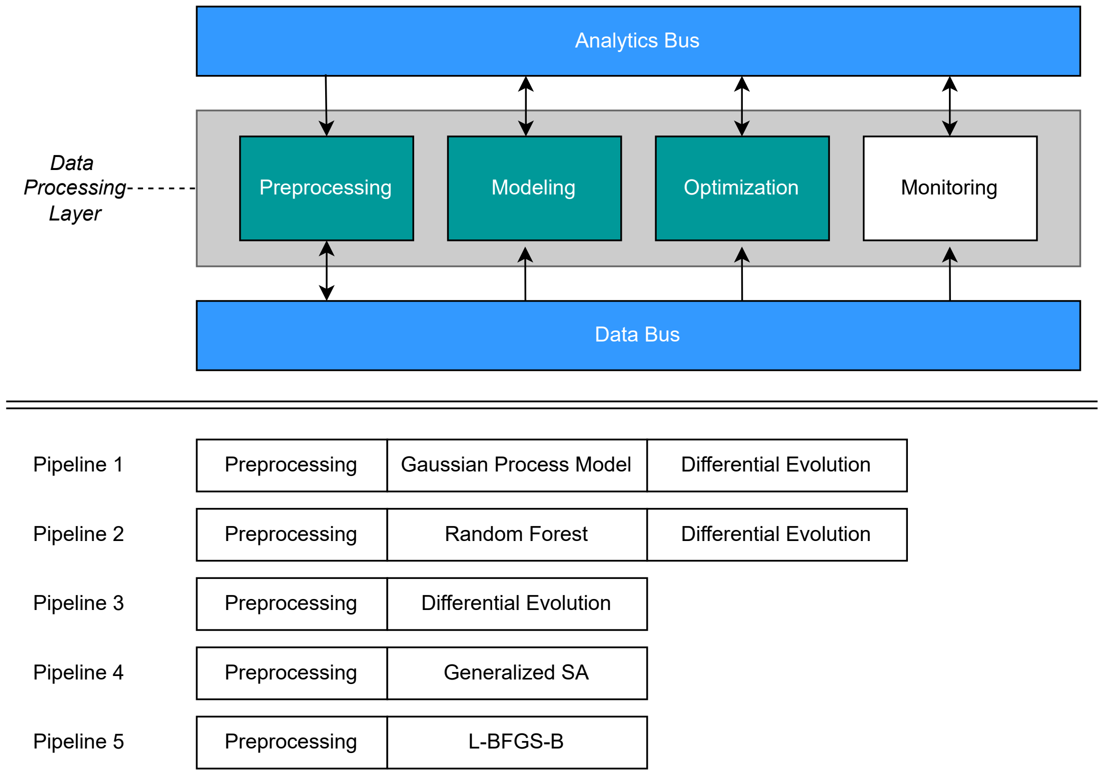

Figure 5.10: Top: *DPL with modules highlighted that can be selected and varied by the cognitive module.* Bottom: *An example of five different process pipelines. All pipelines use the same (shared) results of one preprocessing module. Two different modeling algorithms are trained on the preprocessed data (Gaussian Process Model and Random Forest) and searched for an optimum by Differential Evolution (pipelines 1+2). Three pipelines use an optimizer directly after the preprocessing (pipelines 3-5): Differential Evolution, Generalized SA, and L-BFGS-B. This represents a typical example, where the cognition module selects algorithms from different families for a continuous optimization problem.*

(ii) Modeling modules receive pre-processed data from the *data bus* and send their results to the *analytics bus*.

(iii) Model application modules receive data, e.g., the trained model, from the *data bus* and the *analytics bus* and send their results to the *analytics bus*.

(iv) Optimization modules receive data from the *data bus* and the *analytics bus* and send their results to the *analytics bus*.

The modules are instantiated with their configuration retrieved from knowledge via the analytics bus. The modeling modules contain various machine learning algorithms in a modular fashion. There is a need for the architecture to incorporate multiple algorithms to select appropriate modules based on the task, the type of data, and the resulting model. As a result, multiple modules can be processed in parallel, see Fig. 5.11 for an example. Both modeling algorithms in this example (Random Forest and Kriging)

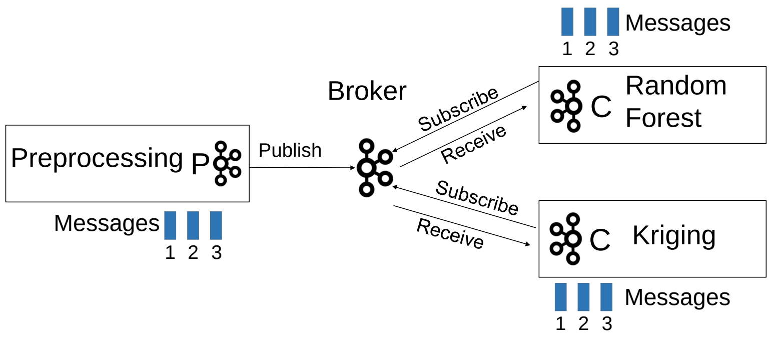

Figure 5.11: *Parallel processing of messages through different algorithms. The cognition module instantiated two pipelines with candidate algorithms and assigned them to different consumer groups. As two consumer groups are subscribed to this topic, both groups receive all new messages, and various algorithms can be trained independently. More instances can be added to the same consumer group, which results in a distributed processing of incoming messages.*

share the same results from the previous preprocessing. Model application modules can access the final model on the *analytics bus*. In addition, the model application will access the *data bus* to compare the model with the process data to detect deviations, which will be provided to the *analytics bus*. The cognitive module ensures that each model application module is compatible with a specific task and a specific model. Each of these components has a specific purpose, such as condition monitoring, predictive maintenance, diagnostics, optimization, or similar tasks. Note that the task of parameter optimization does not specifically require modeling. An example of five different pipelines for a continuous optimization problem with algorithms selected from the algorithm families discussed in Sec. 3.1 is shown in Fig. 5.10. The first two pipelines consist of three subsequent modules (preprocessing, model, and optimizer), and the last three pipelines optimize the CPPS directly evaluating the preprocessed data.

Conceptual Layer The Conceptual Layer (CL) is located between the *analytics bus* and the *knowledge bus* and contains the following four modules.

(i) The reporting module visualizes the process data for the Human-Machine Interaction (HMI). It processes data resulting from, e.g., monitoring or model application results.

(ii) The knowledge module includes

(a) relevant information about the CPPS, such as signal names, device types, or its topology,

   (b) general knowledge, such as an algorithm topology that describes the capabilities and properties of algorithms, and
   (c) constraints that can be defined by the user, such as time constraints.

(iii) The business logic module decides whether or not an action is required. To do this, it monitors the results from the model application modules, checks the constraints from the knowledge module, and derives actions, such as adjusting the CPPS to set optimized process parameters.

(iv) The cognition module is responsible for creating and optimizing the processing pipelines. When given a specific task by the user, the cognition module aggregates and configures appropriate modules of the DPL into processing pipelines by processing the algorithm topology of the knowledge module. By monitoring the results of the pipelines and switching to more promising and better performing pipelines, performance can be improved over time. Therefore, the cognitive module is an elementary module of the CAAI and the reason why we call it a cognitive architecture.

5.3.5 Cognition

The cognition module is a crucial part of the CAAI architecture, as it enables the system to learn over time and transfer knowledge to multiple use cases (G-3). It is responsible for important tasks in the CAAI architecture, such as algorithm selection and processing pipeline generation, parameter tuning, and performance monitoring of the CPPS. To properly address these tasks, the following prerequisites must be met:

(i) Feature engineering is the task of selecting and extracting relevant features from sensor data after or during preprocessing. Involving domain knowledge and experience of the engineers is considered a prerequisite and can significantly speed up the process time and increase the quality of the resulting models and applications.

(ii) A declarative goal for the system must be given, e.g., ”minimize energy consumption.” Furthermore, the goal must be reflected in the CPPS and the sensor data. In Sec. 4.1 we presented a multi-stage selection process using available sensors and parameters of the CPPS. A set of appropriate algorithms must be available to address the specified goal.

(iii) Finally, relevant knowledge and business logic must be available to solve the given task and adapt the CPPS.

The cognitive module selects feasible processing pipelines as candidates to meet the configured goal. Pipelines consist of, for example, preprocessing, modeling, model applications (such as classification, regression), or optimization. While some modules may require specific predecessors, e.g., a particular preprocessing, other module instances may be the same across pipelines, so their results need only be computed once. The CBDP orchestrates the sequence of modules and manages their processing, which can be in parallel. Once the pipelines are selected, the initialization step involves tuning the hyperparameters to avoid incorrect configurations and parameter settings that lead to poor performance. Finally, the cognition evaluates the candidate pipelines and selects the best according to their performance. An algorithmic procedure for the goal of continuous parameter optimization is shown in Algorithm 1 in Sec.4.2. This algorithm can be used as a blueprint for other use cases.

5.4 Conclusions

In this chapter, we have specified requirements gathered from several use cases in the context of CPS and defined goals (G1-G4) to be achieved by a cognitive architecture to improve or maintain the efficiency of CPPSs. Each goal is addressed by a particular method (M1-M4) that can be implemented by multiple technologies or solutions, e.g., (T1)-(T5). Figure 5.12 details the coherence of these goals, methods, and solutions, resulting in our cognitive architecture CAAI, which has been presented and will be further evaluated on real-world problems in the following chapter. A key feature is the cognition module, which configures, instantiates, and evolves process pipelines over time to solve the problem, i.e., to achieve the declaratively configured goal by the operator. This is done with a minimum of expert knowledge and no manual implementation effort.

The presented architecture provides horizontal and vertical scalability, as modules can be run in parallel and deployment on high performance workstations or even cloud infrastructure (local or remote) is considered. This allows its use in many scenarios, especially in the context of big data. Different optimization use cases, e.g. multiple machines, can be solved even without changing the implementation. From our point of view, the architecture can be classified as a system architecture with reference character, due to the described goals and methods. The solutions, technologies, and implementation blueprints and prototypes, such as the cognitive module and the specified algorithm portfolio, provide detailed guidance towards a software architecture. This directly answers research question **RQ-3**, since the benefit of the architecture increases with less abstraction while maintaining generalization in terms of the use cases addressed. In the next chapter, we will examine prototypical implementations for several real-world use cases to evaluate the architecture and the performance of the cognition module with respect to the optimization problems.

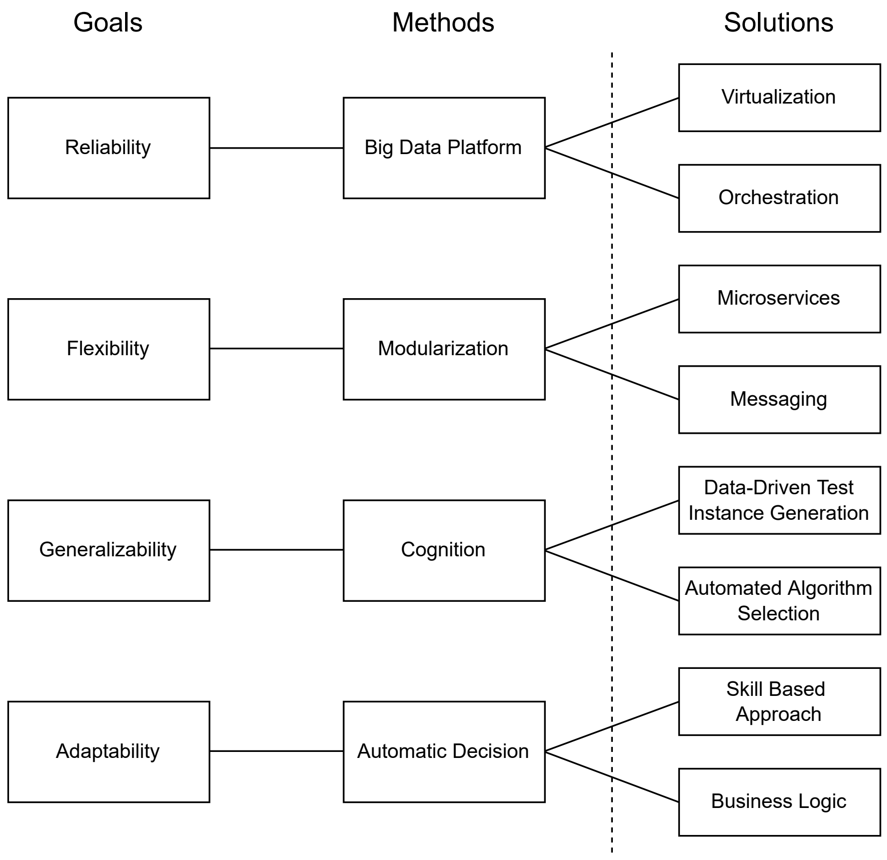

Figure 5.12: *Overview of the architecture's goals, methods, and solutions. The goals and methods on the left represent the higher-level reference parts of the architecture, while the solutions on the right represent the current state of the art or promising approaches in their respective disciplines.*

Chapter 6

Evaluation

This chapter contains work and results based on the following publications [53, 146, 52]. The notation and structure was adapted to this thesis, some parts of the text were taken verbatim.

To evaluate the architecture and the developed algorithm selection procedure, we will present two different real-world case studies. The first case study in Section 6.1 examines the performance on the Versatile Production System using two different algorithm portfolios, and the second case study in Section 6.2 demonstrates the parameter optimization of the injection molding process. A concluding summary is given in Section 6.3.

6.1 Case Study 1: Versatile Production System

We evaluate the CAAI, which is presented in Sec. 5.3, through its implementation for the VPS, which is located in the SmartFactoryOWL. There are typically four VPS units: Delivery, Storage, Dosing, and Production (see Fig. 5.1). Due to its modularity, the first three units can be easily swapped or removed. Different configurations are used depending on the current orders. The need for different configurations increases when, for example, the dosing unit needs to produce a small and precise amount of popcorn. However, when larger quantities are required, it is more efficient to dispense with the dispenser, which is slow and expensive to operate. In this use case, all VPS units are used and small boxes of popcorn are produced. Each batch must fill one box of popcorn. The excess popcorn produced in a batch, or boxes that are not fully filled, cannot be used, so it is waste.

6.1.1 Multi-objective Optimization

A multi-objective optimization problem can be defined by a function

$$f : A \subset \mathbb{R}^n \to \mathbb{R}^m, f(x) = (f_1(x), ..., f_m(x)) \tag{6.1}$$

with $x \in \mathbb{R}^n$, and the feasible set A. A common way to handle multi-objective optimization problems is Pareto optimization based on the concept of Pareto-dominance. A solution $x \in A$ dominates ($\prec$) another solution $y \in A$:

$$x \prec y \quad \text{iff} \quad \forall i : f_i(x) \leq f_i(y) \wedge \exists j : f_j(x) < f_j(y) \quad \text{,for} \quad i, j \in \{1, ..., m\} \tag{6.2}$$

The set $\{x \in A \mid \nexists y \in A : \; y \prec x\}$ is called the Pareto-set, and the corresponding set under mapping f is called the Pareto front.

6.1.2 Optimization Problem

The goal of this use case is to optimize the amount of corn in the reactor as delivered by the dosing unit. The optimum is a trade-off between three minimization functions with conflicting goals: the energy consumption (f_1), the processing time (f_2), and the amount of corn needed for a small box (f_3). The result of the optimization is a parameter value for the dosing unit. To address this multi-objective optimization problem, at least two promising approaches can be considered: An approach using multi-objective optimization algorithms, or an aggregation of the three objective functions and the implementation of a single-objective optimizer to find a solution. To keep the focus on the evaluation of the developed concepts and the architecture, we aim at keeping the complexity of the problem as simple as possible. Consequently, we apply a single-objective approach and compute a weighted sum of the objectives.

This leads to the following optimization problem.

$$\arg\min \sum_{i=1}^{3} w_i f_i(x); \quad \text{w.r.t} \;\; x \in [3000, 11000] \text{ and } w_i > 0 \text{ and } \sum_{i=1}^{3} w_i = 1 \tag{6.3}$$

The parameter x controls the runtime of the conveyor (in ms) and thus affects the amount of corn processed, the cycle-time, and hence the total energy consumption per cycle. The scalar weights of the corresponding objectives, w_i, are chosen based on the user's preferences. We use equal weights for each objective, and compute a normalization of the objectives before aggregation. The minimum of (6.3) is a always a Pareto optimal solution (see e.g. [46]), but with this approach, we can only find Pareto optimal solutions on the convex parts of the Pareto front [35]. Consequently, the solutions found by the single-objective approach may be worse than those found by the multi-objective approach, depending on the shape of the true Pareto front.

A single-objective approach improves the performance of the system with a certain effort in terms of the number of objective function evaluations, i.e., the number of production cycles, and does not depend on a human decision maker, but comes with the trade-off of solutions that may not be Pareto optimal. Consequently, the quality of the solutions is implicitly determined by the a priori choice of the weights. Without knowledge of the Pareto front, it is not known whether this choice of weights is desired.

In the following, we present two different solutions with different portfolios of algorithms used to optimize the VPS. We will use a fixed-budget benchmark scenario, since the objectives of the optimization problem are not yet known. The first solution is motivated by the Automated Machine Learning (AutoML) approach, an automated process of applying machine learning models to real-world problems, which aims to allow non-experts to use machine learning models [37]. We use two different regression models to approximate the objective function based on some sample data points of the process, and optimizers to search the models for an optimum, which is then validated on the real objective function. This allows the models to be compared and the best model to be selected for further iterations of the process in terms of optimization. The second solution extends the algorithm portfolio by adding optimizers and allowing pipelines without surrogate models, i.e., pipelines that use optimizers to directly optimize the parameters of the function.

6.1.3 Solution 1: Surrogate Model-Based Optimization

In this approach, the problem is optimized by SMBO [73]. SMBO uses a data-driven surrogate model to create an approximation of the real VPS production process. Model construction requires training data, which ideally should be a representative set of all possible settings, i.e., space-filling. In this case, the set is generated by evaluating an equidistantly spaced design in the full parameter range of x. The cognition module will evaluate different surrogate models: Random Forest [22] and GPM, in this context further referred to as Kriging [82]. Kriging is particularly suitable for modeling continuous data with few variables and comes with an uncertainty measure. Random forest is known for its fast computation and flexibility. Recent examples of Kriging and Random Forest applications in CPPS scenarios can be found in [78, 152].

These two modeling algorithms already cover a wide range of systems. The cog-

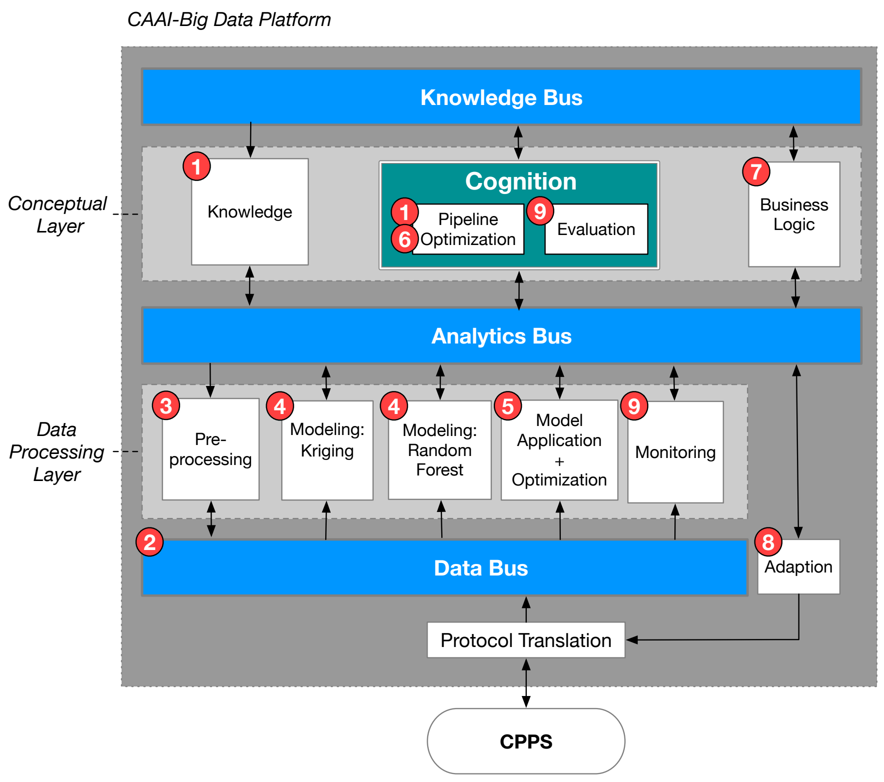

Figure 6.1: *The resulting CAAI architecture for the VPS use case. The numbers indicate the order of the workflow, where some steps can be computed in parallel, i.e., the two different surrogate models.*

nition module decides which model is best suited to approximate the process data and perform optimization based on a performance evaluation of the entire optimization cycle. The surrogate is then optimized to identify the next candidate solution to be evaluated on the VPS by applying a local search algorithm.

When a new parameter is identified, the business logic defines if an adjustment should be performed and checks for constraints, such as parameter limits to ensure a safe operation of the VPS. Finally, the adaption module changes the VPS parameters to achieve better performance.

The resulting implementation of the CAAI for the given use case, including all applied modules and their processing order, is shown in Figure 6.1. The individual steps of the workflow are as follows:

1. The *cognition* module initializes the two pipelines for parallel processing. The knowledge module provides the necessary information about the algorithms and bounds for the control parameter.

2. The *protocol translation* module transfers the data from the OPCUA server on the VPS to the BDP.

3. The *preprocessing* module aggregates the raw data for each production cycle, i.e.

it cumulates the energy consumption, the amount of corn used, and calculates the cycle time. For model training, we also perform data normalization.

4. In this step, the *Kriging* and *Random Forest* models fit or update their parameters to the data and send the results to an analytics data bus for further processing.

5. The *model application* + *optimization* module implements the sequential step of the SMBO algorithm: It searches the previously fitted models until an optimal solution is found or the maximum number of iterations is reached. The module passes the result to the analytics bus.

6. The *cognition* module decides which pipeline prediction is selected to optimize the CPPS by comparing model accuracy using cross-validation.

7. The *business logic* module checks whether the solution violates any constraints defined for the CPPS and communicates the appropriate adjustment back to the analytics bus.

8. The *adaption* module translates the adjustment for the specific CPPS and sends the instructions from the BDP to the CPPS using the *protocol translation*, i.e. writing the new value for the control parameter x to the OPCUA server.

9. The *cognition* module analyzes the system performance achieved with the resulting pipeline configuration from step 6. In the following steps, the impact of the changes is verified and made available to the operator through information provided by the *monitoring* module.

Results

Data from the real-world VPS was collected to evaluate the modeling and optimization. This data consists of 36 production cycles with 12 different settings (three pure repetitions per configuration) for the runtime of the conveyor. Based on this data, we trained a model that reflects the real behavior of the VPS and use it for further experiments. The three different objectives, i.e., the energy consumption, the processing time, and the amount of corn required, were aggregated by taking the sum of the single objectives multiplied by equal weights of $1/3$.

The statistical software R version 3.4.3 was used to evaluate the algorithms [127]. For Kriging and Random Forest, the software packages SPOT (2.0.5) and caret (6.0-84) [84] were used. Initially, the algorithms used five equidistant data points to build their initial models. Consequently, the results in the figures presented in this section start at production cycle number five. The aggregated results use the median of ten repetitions, each with a budget of 20 production cycles.

The obtained values of the objective function are shown in Figure 6.2. Initially, Kriging outperforms Random Forest converging to an optimum after 10 cycles, while later, after 12-15 cycles, Random Forest performs comparably to Kriging.

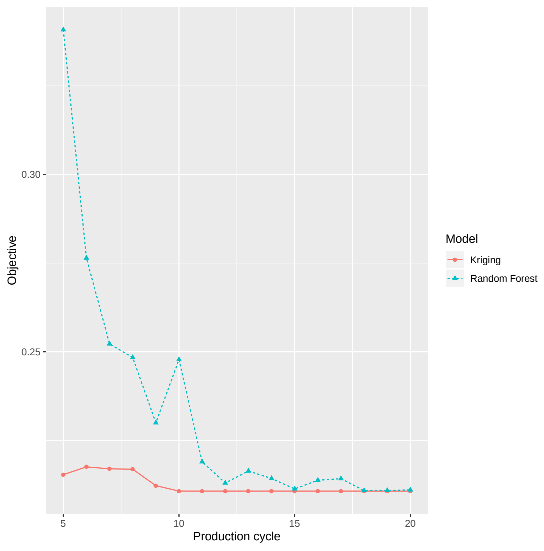

Figure 6.2: *Objective function value plotted against the number of production cycles. Smaller values are better.*

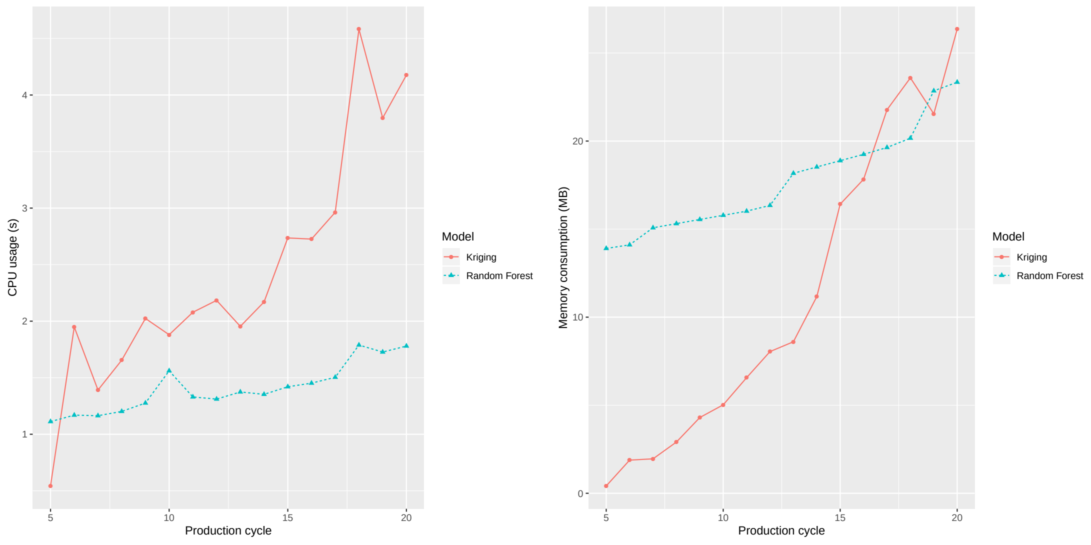

Figure 6.3: *This plot shows the CPU and memory consumption over time using median aggregation over 20 repetitions.* Left: *The consumed CPU time in seconds is depicted over the production cycle.* Right: *The memory (MB) consumption of the modeling algorithms for different production cycles is shown.*

Fig. 6.3 plots the CPU consumption in seconds and the memory consumption in MB against the VPS production cycles. For both methods, Kriging and Random Forest, an increasing trend in the CPU usage can be observed. However, the computation time of Kriging shows a larger slope compared to Random Forest. Both algorithms behave as expected due to their internal data representation and processing. The increasing trend can also be observed for the memory consumption. Initially, the Random Forest algorithm uses more memory than Kriging. After about 15 iterations, the situation changes as Kriging begins to consume more memory than Random Forest. While both algorithms continue to consume more memory, Kriging shows the steeper slope.

The results of our study reveal the following valuable insights:

(i) It is worth using more than one algorithm: taking only the best performing one (i.e. Kriging) can lead to future problems due to limited computational resources and time.

(ii) Random Forest needs more data to improve compared to Kriging, and starts to be a good competitor after about 15 cycles.

(iii) Overall, it may be beneficial to switch algorithms after a certain number of production cycles, considering all performance metrics together, i.e., the achieved objective value, CPU time, and memory consumption, according to user preference and system capacity. Otherwise, the memory and CPU consumption of Kriging can quickly make its use infeasible in terms of available computing resources.

The cognition module uses the aggregated information shown in figures 6.3-6.2 to select the best algorithm for the next production cycles. This also demonstrates the relative degree of information density on the analytics bus compared to the data bus (see Table 5.3), as the cognition module does not use the raw data, which consists of several hundred rows of data for each production cycle.

6.1.4 Solution 2: Extended Algorithm Portfolio

The AutoML-like comparison of multiple algorithms is not possible when using non-SMBO optimizers. The algorithm selection in this solution is performed by comparing several optimizers from different families (see Sec. 3.1) on generated test instances (see Sec. 2.2) using a fixed-budget approach and statistical comparison (see Sec. 3.2). The test instances are generated using 1-stage, conditional GPS with the parameter Ncos set to 100 (see Sec. 2.2.4 for details). For the statistical analysis of the algorithm performance after tuning, we chose a nonparametric test that can be easily run automatically. The generated test instances used for tuning and benchmarking the algorithms are shown in Fig. 6.4. We use the following portfolio of algorithms: Differential Evolution, Kriging, Random Forest, BOBYQA, and Random Search as baseline comparator. For an overview of all algorithms used in this scenario and their parameters, see Tab. 6.1. Tuning is performed using SPOT with the following settings: SPOT's budget is set to 30 evaluations, i.e., algorithm runs, and the algorithm parameters listed in Tab. 6.1 have been selected for tuning. The budget for the algorithms is set to a maximum of 20 function evaluations. SPOT uses a Kriging model for tuning and an initial design size of ten points. The resulting parameter values are also listed in Tab. 6.1. The results of the tuning procedure are shown in Fig. 6.5. It can be seen that Differential Evolution and Random Search are outperformed by the other algorithms and could

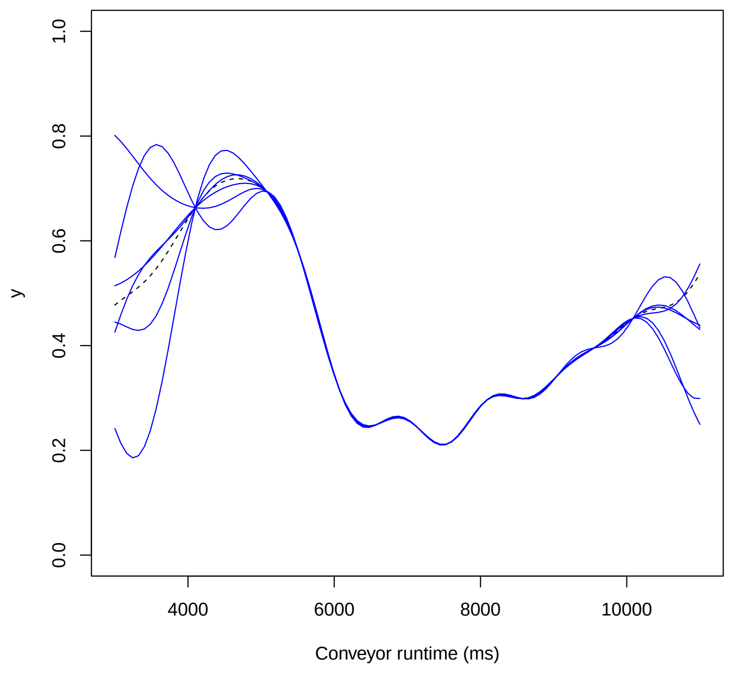

Figure 6.4: *This graph shows the resulting VPS problem instances based on the real-world data taken from the machine. The dashed line shows the ground truth, and the solid blue curves represent the conditional simulations used as test instances for tuning and benchmarking of the algorithm pipelines. The x-axis shows the runtime of the conveyor in ms, and the y-axis shows the equally weighted normalized aggregated objective function value of the objectives process time, amount of corn, and energy consumption.*

therefore be removed from further analysis. However, our fully automated analysis does not rely on figures or plots to remove individual algorithms, but will proceed with a full statistical comparison. The questions to be answered by the benchmark are

- Q-1 Do the algorithms differ in performance after the tuning procedure with a given confidence, and which algorithm performs best?
- Q-2 How much improvement does the best algorithm provide over the baseline?

The statistical analysis to evaluate possible differences between the performance of the algorithms was performed as follows (see Sec. 3.2). First, a Kruskal-Wallis test was performed, which yields a very low p-value ($< 2.2e-16$). Therefore, significant differences between the performance of the algorithms were analyzed in a post hoc test. The results of the multiple pairwise comparison using the Conover's test are shown in Tab. 6.2. The null hypothesis states that any observed differences in the rank sums of the performance of the algorithms are due to chance alone. If the p-values (< 0.05) indicate a significant difference, we want to decide which algorithm is

Table 6.1: *Settings of parameter ranges and corresponding default and tuned values of chosen optimizers*

Parameter	Range	Default	Tuned	Family
Differential Evolution				Population
popsize	$\mathbb{N}_+$	5	4	
strategy	$\{1,2,3,4,5\}$	2	4	
F	$[0,2]$	0.8	1.647	
CR	$[0,1]$	0.5	0.141	
c	$[0,1]$	0.5	0.430	
Kriging (SPOT)				Surrogate
designSize	$\mathbb{N}_+$	7	4	
designType	$\{Lhd, Uniform\}$	Lhd	Lhd	
Random Forest (SPOT)				Surrogate
designSize	$\mathbb{N}_+$	7	5	
designType	$\{Lhd, Uniform\}$	Lhd	$Uniform$	
BOBYQA				Hill-climber
Random Search				Baseline

Table 6.2: *Results of pairwise post-hoc analysis using the Conover test. A significant difference at a significance level of* 0.001 *is marked with *** and indicates that the null hypothesis stated in the first column should be rejected.*

Null hypothesis	q value	Pr(>|q|)	
Differential Evolution - BOBYQA == 0	18.604	2.4425e-13	***
Kriging - BOBYQA == 0	7.138	7.4403e-05	***
Random Forest - BOBYQA == 0	10.004	7.7765e-08	***
Random Search - BOBYQA == 0	20.459	2.4092e-13	***
Kriging - Differential Evolution == 0	-11.466	2.3796e-09	***
Random Forest - Differential Evolution == 0	-8.599	2.3031e-06	***
Random Search - Differential Evolution == 0	1.855	0.68556	
Random Forest - Kriging == 0	2.866	0.27010	
Random Search - Kriging == 0	13.321	3.2829e-11	***
Random Search - Random Forest == 0	10.454	2.6413e-08	***

superior. We select the superior algorithm based on the rank sum comparison between the two algorithms. To obtain a final ranking of the algorithms, we proceed as follows. First, all algorithms that are not significantly worse compared to other algorithms are given the first rank and removed from the list. Each of the remaining algorithms that is not worse than any other of the remaining algorithms is given rank two and removed

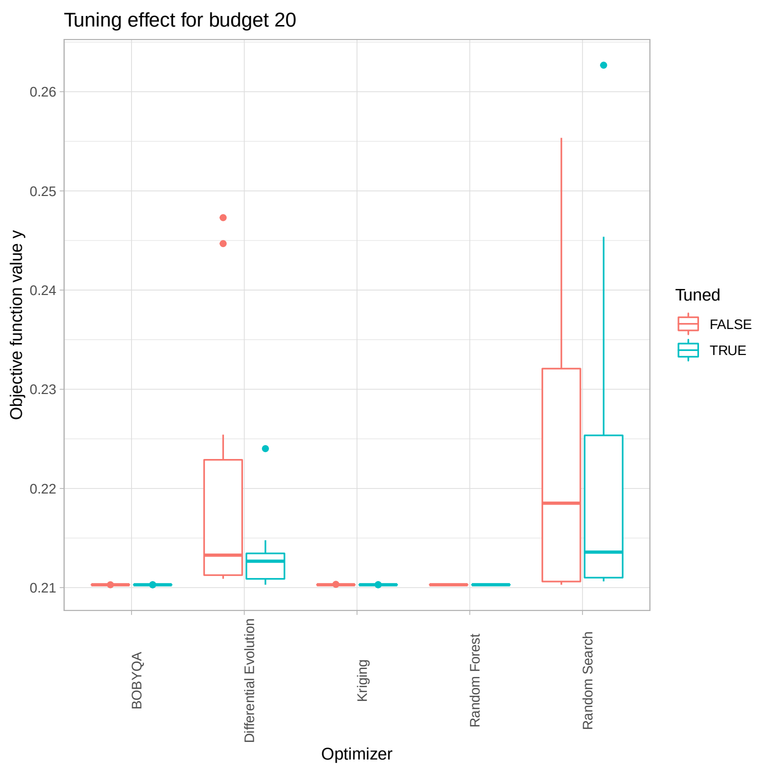

Figure 6.5: *This graph shows the tuning results of all optimizers. It can be seen that Differential Evolution shows a more robust performance after tuning. For Kriging and Random Forest, the size of the initial design was tuned, but the differences in performance are not noticeable. For BOBYQA and Random Search, there were no parameters to tune, so performance differences are only due to chance.*

from the list. This is repeated until all algorithms have received a rank. The analysis of the multiple pairwise comparisons based on the Conover test and the sorting of the ranks using the rank sums yields the final ranking of the algorithms: BOBYQA (1), Kriging (2), Random Forest (2), Differential Evolution (3), Random Search (3). Since only one algorithm receives the first rank, selecting BOBYQA is straightforward. If there are more algorithms with the first rank, we use the average CPU consumption over the repetitions as a secondary objective. With these results, we can answer the two questions above:

Q-1 The difference in the performance of the algorithms on the VPS problem is statistically significant in a majority of the pairwise comparisons. BOBYQA was selected as the superior algorithm based on the significant differences and the rank sums.

Q-2 BOBYQA achieves a relative improvement in VPS performance of 5.65% over the baseline comparator (Random Search).

6.2 Case Study 2: Injection Molding Optimization

The previous case study was used to demonstrate the concept of online algorithm selection based on process data on a rather simple optimization problem with only one control parameter. Even this problem motivated the need to address multiple, possibly conflicting, objectives simultaneously, which is arguably typical for industrial processes. Often, these objectives include the quality of a product or process, the process or cycle time, and the consumption of resources such as a raw material or energy. In this case study, we analyze the extension of the developed automatic online algorithm selection to multi-objective optimization problems by using an appropriate performance metric and an appropriate algorithm portfolio.

6.2.1 Performance Indicator and Algorithm Portfolio

To evaluate the approximation of the Pareto front, achieved by a multi-objective optimizer, we will use the value of the dominated hypervolume as the performance indicator. We will compute the hypervolume of the achieved Pareto set of the final population of the algorithms with respect to a given reference point as a scalar value to evaluate the set of solutions of an algorithm. Maximizing this hypervolume pushes the solutions towards the desired objective values and rewards a higher diversity of solutions along the border to the non-dominated region. Alternatives for the metric to compare the performance of algorithms are, e.g. the Inverted Generational Distance (IGD) [30] or Inverted Generational Distance Plus (IGD+) [69]. These alternatives assume that the Pareto front or an approximation of the Pareto front is already known, so they are not feasible for our black-box scenarios.

Well-known Multi-objective Evolutionary Algorithms (MOEAs) such as NSGA-II, SPEA2, SMS-EMOA, MOPSO, and MOEA/D have become standard solvers for multi-objective optimization problems [50]. MOEAs generalize the idea of EA and are typically designed to iteratively approach solutions that are well distributed across the Pareto front. Exploiting a population of individual solutions that collectively approximate the Pareto front harmonizes well with processes and concepts in natural evolution [50].

Three main paradigms for the design of MOEAs can currently be distinguished [50]:

- Pareto-based MOEA, where the algorithms use a two-stage ranking scheme. First, the Pareto dominance properties of individuals are used to determine the ranking. If a decision between equally ranked individuals is required, a second level ranking is performed using properties contributing to diversity.

- Indicator-based MOEA, where the algorithms measure the performance of a solution set according to an indicator such as the hypervolume indicator. This measure is used by the algorithms in such a way that improvements in its value determine the selection or ranking procedure of its individuals.

- Decomposition based MOEA, where the algorithms decompose the problem into several subproblems. Each subproblem targets different parts of the Pareto front and uses a different weighting of a scalarization method.

Consequently, our algorithm portfolio for multi-objective optimization problems consists of the following MOEAs algorithms: S-Metric Selection Evolutionary Multiobjective Optimization Algorithm (SMS-EMOA) [15] (indicator-based), Nondominated Sorting Genetic Algorithm III (NSGA-III) [39, 70] (decomposition-based), and Multiobjective Evolutionary Algorithm based on Decomposition (MOEA/D) [157] (decomposition-based).

SMS-EMOA uses the hypervolume indicator to evaluate the quality of a Pareto front approximation. Its main idea is to explicitly aim at maximizing the hypervolume indicator within the optimization process [15]. It requires a predefined reference point for calculating the size of the hypervolume.

MOEA/D decomposes a multi-objective optimization problem into a number of scalar optimization subproblems for simultaneous optimization. It uses evolutionary techniques to find optimal solutions. The subproblems are the individuals that represent the population of the algorithm. Each individual is assigned a unique weight vector. The weights are chosen to be equally distributed among the different objectives. On the set of subproblems, a neighborhood can be defined such that good solutions in one subproblem should be close to good solutions in neighboring subproblems due to their proximity and thus similarity in the fitness landscape. Thus, the algorithm can update neighboring individuals by comparing their objective values.

NSGA-III uses reference points on a normalized hyperplane to guide the evolution of its individuals towards the Pareto front. Solutions are associated with the reference points, aiming for a uniform distribution. We use the Das-Dennis approach to compute the reference points [36]. The NSGA-III uses the same non-dominated sorting procedure as in NSGA-II.

Please note that this portfolio is easily customizable and is not intended to solve all existing problems in the most efficient way. We chose these algorithms because they have shown good results on many problems and because they implement different approaches to approximate the Pareto front. For our solution, we use the implementation of the algorithms from the *pymoo* package [18] (version 0.6.0.1) for the Python platform (version 3.9).

6.2.2 Multi-objective Optimization Problem

In this case study, we want to optimize the injection molding filling simulation. Therefore, we will use the three largely uncorrelated objectives of the injection molding simulation already discussed in Sec. 2.3.1: The average volume shrinkage [%] (f_1), the maximum warpage [mm] (f_2), and the required cooling time [s] (f_3). Consequently, the multi-objective optimization problem to be solved can be formulated as follows:

$$\min_x f(x) = (f_1(x), f_2(x), f_3(x)), x \in \mathbb{R}^3 \tag{6.4}$$

Table 6.3: *Lower and upper boundaries for the control parameters of the injection molding simulation optimization problem*

	x_1	x_2	x_3
lower	230	0.3	10
upper	250	6.3	50

Table 6.4: *Default parameters of the multi-objective evolutionary algorithms*

	μ	p_c	p_{vc}	η_c	p_m	η_m	n_n	p_{nm}
SMS-EMOA	100	0.9	0.5	15	0.9	20		
NSGA-III	100	0.9	0.5	15	0.9	20		
MOEA/D	100	0.9	0.5	15	0.9	20	20	0.9

Table 6.5: *Lower and upper boundaries for the parameters selected for tuning*

	μ	p_c	p_{vc}	η_c	p_m	η_m	n_n	p_{nm}
lower	40	0.5	0.1	3	0.7	3	3	0.1
upper	150	1.0	0.9	30	1.0	30	20	0.9

The lower and upper bounds for x are given in Tab. 6.3, and x_1 is the melt temperature [°C], x_2 is the holding pressure time [s], and x_3 is the cooling time in the mold after filling [s].

6.2.3 Experiments and Tuning

The default parameters of the algorithms are listed in Tab. 6.4. These parameters are the population size μ, the crossover probability p_c, the probability for each variable to participate in crossover p_{vc}, the η parameter for crossover η_c, the mutation probability p_m, the η parameter for mutation η_m, the number of neighbors n_n, and the probability for neighbor mating p_{nm}. Note that the last two parameters are only valid for MOEA/D.

We apply a tuning procedure with the goal of optimizing the control of the algorithms to maximize the dominated hypervolume after a certain number of function evaluations. Changing the value of μ for MOEA/D requires the set of reference points to be of equal size, making the Das-Dennis approach for generating well-spaced reference points infeasible. Instead, we use a so-called s-energy approach, which allows the setting of an arbitrary number of points [19]. This allows tuning of all listed parameters. The lower and upper bounds for tuning are given in Tab. 6.5.

The test instances are generated per objective using 1-stage, conditional GPS with $Ncos$ set to 300. The data is based on an initial *Lhd* sampling of the Cadmould 3D-F simulation with 20 points using the lower and upper boundaries of the control parame-

Table 6.6: *Resulting parameter values for the multi-objective evolutionary algorithms after tuning*

	μ	p_c	p_{vc}	η_c	p_m	η_m	n_n	p_{nm}
SMS-EMOA	77	0.70	0.87	21	0.91	10		
NSGA-III	97	0.63	0.24	14	0.89	26		
MOEA/D	150	0.79	0.90	23	0.88	3	3	0.47

ters listed in Tab. 6.3. All algorithms were allowed to optimize the objective functions with a maximum of 2000 function evaluations and 10 repetitions. SPOT was used to tune the algorithms with a budget of 30 evaluations for each algorithm. The reference point for the hypervolume computation was set to $r = (3.5, 60, 0.1)$ for the objectives f_1, f_2, and f_3 respectively. This reference point is also used as a maximum, such that the achieved objective values of the Pareto sets can be normalized between a minimum point $p = (0, 0, 0)$ and the maximum.

6.2.4 Results

The resulting parameter values after the tuning are shown in Tab. 6.6. The achieved hypervolume in the final populations per algorithm run after 10 repetitions before and after the tuning can be seen in Fig. 6.6. Visually, there is no significant difference after tuning recognizable for SMS-EMOA and NSGA-III. However, MOEA/D achieved improved hypervolume values after tuning, but is significantly outperformed by the other algorithms. It will be included in the following statistical analysis anyway, as it is performed automatically.

The questions to be answered by the benchmark are

Q-1 Do the algorithms differ in performance in terms of hypervolume after the tuning procedure with a given confidence, and which algorithm performs best?

Q-2 How much improvement does the best algorithm provide over the baseline?

For the statistical analysis of the algorithm performance after tuning, we chose the same procedure as for the VPS use case (see Sec. 6.1.4). The Kruskal-Wallis test yields a very low p-value ($< 3.482e - 13$). Therefore, significant differences between the performance of the algorithms were analyzed by the multiple pairwise comparison using Conover's post hoc test. The results are shown in Tab. 6.7. The analysis of the multiple pairwise comparisons based on the Conover test and the sorting of the ranks using the rank sums yields the final ranking of the algorithms: SMS-EMOA (1), NSGA-III (2), and MOEA/D (3).

Therefore, SMS-EMOA is used to optimize the problem. In this use case, the computed Pareto front is used by a human decision maker to select a solution according to his subjective preferences based on his expert knowledge. Several solutions can be compared visually at the same time, as shown in Fig. 6.7, an example of such a

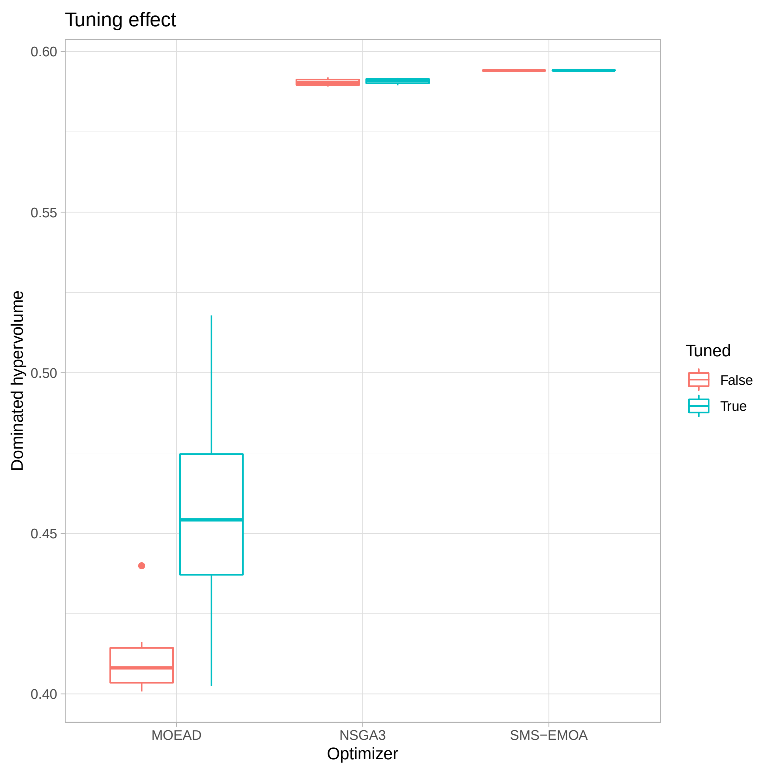

Figure 6.6: *This graph shows the tuning results of all three multi-objective optimizers. There is no significant difference for SMS-EMOA and NSGA-III. MOEA/D achieved a significant improvement in terms of the dominated hypervolume indicator after the tuning procedure.*

Table 6.7: *Results of the pairwise post-hoc analysis using the Conover test. A significant difference at a significance level of* 0.001 *is marked with *** and indicates that the null hypothesis stated in the first column should be rejected.*

Null hypothesis	q value	Pr(>|q|)	
NSGA-III - MOEAD == 0	10.445	1.7849e-07	***
SMS-EMOA - MOEAD == 0	20.889	6.0063e-14	***
SMS-EMOA - NSGA-III == 0	10.445	1.7849e-07	***

Pareto front obtained by SMS-EMOA. The plot of the Pareto front allows the decision maker to visualize the trade-offs between different objectives. It is often not possible to optimize all objectives simultaneously, and the Pareto front shows the tensions and trade-offs between objectives. By examining the Pareto front, the decision maker can choose a solution that best meets his or her preferences. This may be a solution that excels in a particular goal domain, or it may be a balanced compromise between different goals. In the example shown in Fig. 6.7, a single solution from the Pareto front is highlighted with a fairly low value for the required cooling time and a trade-off between the average volume shrinkage and the maximum warpage. Because the Pareto

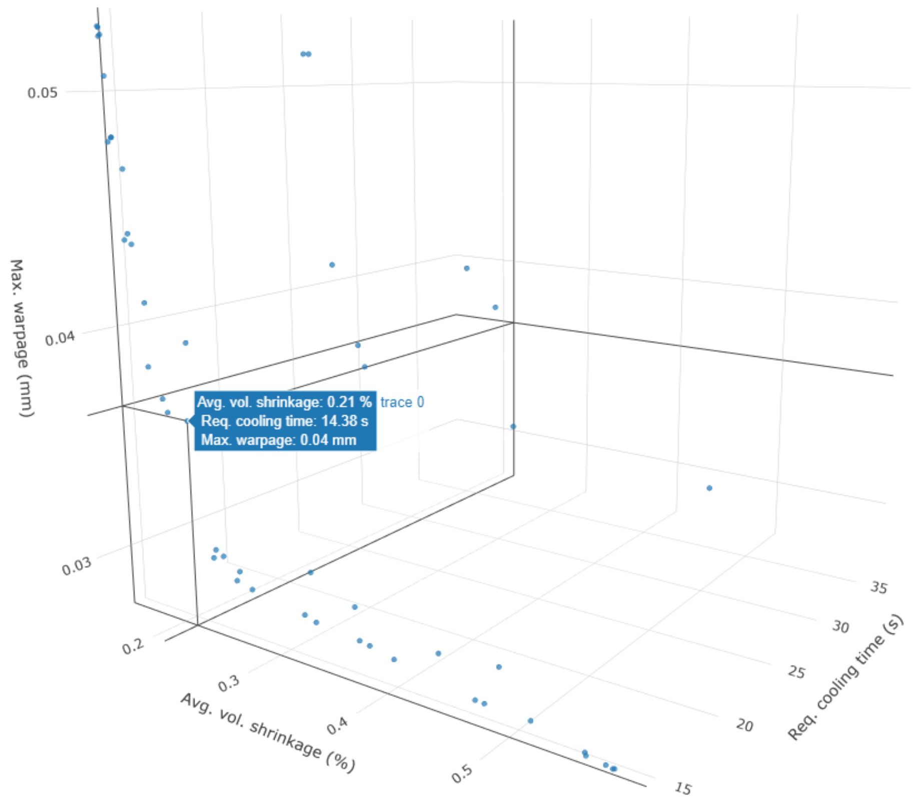

Figure 6.7: *This plot shows a Pareto set computed by SMS-EMOA for the injection molding optimization problem. Each point represents a Pareto optimal solution and the plot can be interactively analyzed, rotated and zoomed by a user, e.g. the human decision maker.*

front illustrates trade-offs between objectives, it allows the decision maker to weigh which combination of objectives best meets requirements or preferences. It facilitates informed decision making by considering different options.

With these results, we can answer the two questions above:

Q-1 The difference in the performance of the algorithms on the injection molding simulation problem is statistically significant in all pairwise comparisons, and SMS-EMOA was selected as the superior algorithm in terms of dominated hypervolume.

Q-2 To achieve a comparable result in terms of the three objectives, we proceeded as follows: Instead of a human decision maker, we specify four different weight vectors $\vec{v}_1 - \vec{v}_4$ with three components each, representing the weight for each objective. The first vector gives equal weight to each objective, the second gives more weight to the first objective, the third gives more weight to the second objective, and the fourth gives more weight to the third objective. The results of the relative improvement over a baseline are shown in Tab. 6.8. Note that such

Table 6.8: *Relative improvement of the injection molding simulation compared to a baseline*

Weight vector	y_3	y_4	y_5
$\vec{v}_1 = (1/3, 1/3, 1/3)$	66.79 %	12.35 %	-45.32 %
$\vec{v}_2 = (0.8, 0.1, 0.1)$	16.34 %	7.28 %	23.68 %
$\vec{v}_3 = (0.1, 0.8, 0.1)$	44.27 %	-2.79 %	9.16 %
$\vec{v}_4 = (0.1, 0.1, 0.8)$	7.86 %	24.77 %	-0.82 %

an automatic procedure may select a worse solution compared to a human decision maker. Overall, the improvement in the required cooling time is noticeable (except for $\vec{v}_3$, with a slightly worse cooling time: -2.79%), which is important since the cooling time determines the cycle time of the process and thus the throughput. The largest improvements can be seen in the average volume shrinkage, where SMS-EMOA achieved results for all four vectors ranging from $7.86\% - 66.79\%$ improvement. It can be seen that a baseline can improve one objective at a time (especially y_5, the max. warpage), but SMS-EMOA optimizes several objectives simultaneously.

6.3 Conclusions

In this chapter, we evaluated the architecture and the automatic algorithm selection approach on several optimization problems. The specification of algorithm portfolios with algorithms from different families is crucial for an automatic approach in the context of black-box problems. On the VPS problem, we could show that a solver with an arguably lower degree of complexity, i.e., BOBYQA, can outperform more complex algorithms, depending on the fitness landscape of the problem, which might be previously unknown. The automatically selected algorithm was able to improve the performance of the VPS by 5.65%, which can be directly translated into energy savings, resource savings, and higher throughput.

When it comes to the application of multi-objective optimizers, we could demonstrate the ability of the cognition module to automatically improve and select algorithms in terms of the dominated hypervolume. We demonstrated the feasibility of the online algorithm selection approach for multi-objective problems on an injection molding optimization problem using filling simulations to estimate the quality of the produced parts and to minimize the cycle time in terms of the required cooling time. In each of the automatically selected solutions, according to four different weight vectors, the solutions obtained by SMS-EMOA were significantly superior to the baseline in at least two of the objectives. The solutions obtained with the highest weight for the maximum warpage achieved a relative improvement of 24.77% in the required cooling time and 7.86% in the average volume shrinkage, while only the maximum warpage was slightly worse (-0.82%) compared to the baseline. Overall, this is a huge improvement and supports the human decision maker in providing a large set of high quality candidate solutions.

The designed architecture and its high degree of automation in algorithm selection drastically reduces implementation time, as different problems are mostly solved by changing configurations. New algorithms can be easily added as needed and are automatically tuned when their control parameters and bounds are configured. The cognition module automatically performs statistical tests after tuning, minimizing uncertainty in the selection process.

Chapter 7

Final Evaluation

7.1 Research Questions and Contributions

At the beginning of this document, we formulated research questions that address several challenges related to automatic algorithm selection based on data-driven test instances for CPPS optimization. To conclude this document, we will discuss the contributions of this work to these challenges.

We asked **RQ-1**: Are GPS and model variations feasible and efficient to allow online algorithm selection, and what are the limits to efficient implementation? To sharpen this question, we asked **RQ-1.1** whether the different objectives, different test instance generation procedures, or different parameter settings of these procedures lead to discernible differences in the instance space, i.e., do they produce different landscape feature vectors? In addition, we asked **RQ-1.2** Do changes in these parameter settings favor or disfavor any algorithm?

This led us to our first contributions:

(C-1) We have experimentally validated the three proposed test instance generation strategies on several injection molding simulation optimization problems. Our results show that existing test instances from the well-known BBOB testbed are not relevant for the considered optimization problems. Therefore, the data-driven approach using GPM was shown to be feasible for the use case.

(C-2) In our experiments, we have shown that it is the objective that determines the performance ranking of the algorithm, and not the strategy for generating test instances. For the online algorithm selection problem for optimizing CPPS, we concluded that the conditional one-stage Gaussian process simulation is the most appropriate.

Since the focus of this thesis is to develop a solution with a high degree of automation, we asked **RQ-2** How can a solution to the ASP be algorithmically implemented, such that operators can optimize a CPPS online with a minimum of data science knowledge and without a hand-written procedure?

(C-3) To map the feasible algorithms to an optimization problem in CPPS, we developed a multi-stage problem configuration procedure. This procedure enables the formulation of optimization problems with a minimum of expert knowledge: The selection of the control parameters and objectives from sources such as OPCUA, i.e., without the need for the knowledge of optimization experts.

To add value, we asked **RQ-3** How can the solution of data-driven online algorithm selection be addressed by a system architecture that provides value to many companies, and what are the requirements for implementation?

(C-4) We have specified a system architecture that describes the necessary components for processing data streams for optimization, and is easily extensible due to its modularity and flexibility. The presented architecture provides horizontal and vertical scalability, as modules can be run in parallel and deployment on microcontrollers, high performance workstations, or even cloud infrastructure (local or remote) is considered. This allows its use in many scenarios, especially in the context of big data. The message-driven approach allows for language-independent implementation, making it easy to adapt to the needs of the companies. This significantly reduces the implementation effort.

7.2 Discussion and Conclusion

In summary, this thesis has contributed significantly to several research areas to advance the degree of automation in CPPS. We developed an automatic solution for selecting optimization algorithms for newly implemented or adapted production processes. This was made possible by evaluating different test instance generation methods based on GPM in terms of ELA and the impact of these methods on the performance ranking of optimization algorithms. The limitations of the approach are closely related to the limitations of using GPM for optimization problems in general: The model must be of sufficient quality, which can be checked using common model validation methods such as cross-validation. New data can be added, for example by applying space-filling designs. If necessary, the implementation of the cognition module can be extended if special design methods are required for the process to be optimized.

The architecture developed enables cost savings in the implementation of production process optimization solutions, especially in the context of large data and data streams. It addresses the shortage of both engineering and automation professionals. The algorithmic implementation strongly supports CPPS operators and potentially increases overall system efficiency by largely automatically optimizing control parameters. To benefit from the solution described, data quality must be sufficiently high. This means ensuring that the relevant machine parameters can be collected with sufficient frequency and accuracy. If the data is coming from a source other than OPC UA, then the protocol translation needs to be extended in such a way that the new data is compatible with a Kafka data stream. The company needs information technology personnel who are familiar with the software and its implementation, and who can configure, customize, and extend it. Therefore, appropriate resources and budget must be considered.

The modular structure of the architecture provides horizontal and vertical scalability, addressing a wide range of companies with different IT infrastructure requirements. Implementations have been tested for microcontrollers and on-premise cluster-based systems. If the optimization problem of the CPPS changes, e.g. if new control parameters are added as a result of process adaptation, the need to change the implementation is largely done by adjusting and extending the configuration of the objective and the required data, since the optimization problems are formulated in a declarative manner.

However, configuring and deploying a multi-container Docker application based on Kafka can be a complex task, and several technical challenges can arise along the way. These challenges often revolve around network configurations, resource allocation, and ensuring proper coordination between different Kafka components. Here are some common technical issues and challenges encountered during development:

Network Configuration: Docker containers within a Kafka application need to communicate effectively. Ensuring proper networking, such as setting up container networks or using Docker Compose networks, is critical for seamless communication between Kafka brokers, producers, and consumers. Configuring the network to allow communication between containers and the host machine is critical. Kafka brokers often require specific ports to be exposed, and ensuring that these ports are accessible while maintaining security is a challenge.

Resource Allocation: Determining the appropriate resource allocation for each Kafka container, including brokers and ZooKeeper nodes, is essential for optimal performance. Incorrect allocation can lead to performance bottlenecks or resource contention. Kafka relies heavily on disk I/O for persistence. Configuring storage volumes, ensuring sufficient disk space, and optimizing disk performance are important aspects of deploying a Kafka cluster in Docker containers.

Dynamic Configuration: Dynamically managing and updating Kafka configurations in Docker containers requires a robust system. Ensuring that configuration changes are applied consistently across the cluster without causing disruption is a challenge.

Security Considerations: Implementing secure communication channels between Kafka components is essential. This includes configuring SSL/TLS for encryption and authentication to ensure that sensitive data is protected during transmission. Properly configuring access controls and authentication mechanisms for Kafka brokers and nodes within Docker containers is critical to preventing unauthorized access.

Monitoring and Logging: Implementing effective monitoring and logging for Kafka containers can be challenging. Tools such as Grafana can be used to monitor container health, Kafka metrics, and log data.

Overcoming these technical challenges requires a thorough understanding of both the Kafkas architecture and Docker containerization. In addition, staying abreast of updates and best practices for both technologies is essential to successfully configuring and deploying a multi-container Kafka application in an industrial context.

We demonstrated the developed solutions on two real-world use cases, where we successfully optimized different single and multi-objective optimization problems. The specification of a big data architecture with cognitive capabilities in terms of automatic algorithm selection and thus optimization of CPPS is a significant step towards fully self-optimized production systems. The extension of the developed concepts and solutions to additional applications, e.g., predictive maintenance or condition monitoring, is the next logical step towards such a system.

List of Figures

List of Tables

Glossary

ACT-R Adaptive Control of Thought-Rational. 14, 78, 79

AI Artificial Intelligence. 81–83

ANOVA Analysis of Variance. 47, 48, 55

ASP Algorithm Selection Problem. 8–12, 16, 17, 42, 66, 113

AutoML Automated Machine Learning. 96

BBOB Black-Box Optimization Benchmarking suite. 17, 34

BDP Big Data Platform. 63–65, 82, 97, 98

BFGS Broyden-Fletcher-Goldfarb-Shanno. 44

CAAI Cognitive Architecture for Artificial Intelligence in Cyber-physical Production Systems. 68, 81–84, 88, 91, 92, 95, 97, 118

CBDP CAAI-Big Data Platform. 83–85, 87, 92

CFD Computational Fluid Dynamics. 29, 45

CL Conceptual Layer. 90

COBBS Continuous Optimization Benchmarks By Simulation. 25

CPPS Cyber-physical Production System. 8–13, 16–20, 39, 40, 46, 55, 57, 59, 61, 63–69, 72, 74, 75, 80–84, 87, 90–92, 96, 98, 113–115

CPS Cyber-physical System. 8, 68, 69, 73, 77, 92

DOE Design of Experiments. 43

DPL Data Processing Layer. 88, 89, 91

EA Evolutionary Algorithm. 44, 104

EGO Efficient Global Optimization. 45

ELA Exploratory Landscape Analysis. 9, 27, 33, 38, 39, 42, 114

ERP Enterprise Resource Planning. 77

flacco Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems. 31

GenSA Generalized Simulated Annealing. 49, 51

GLG Gaussian Landscape Generator. 18, 49–52

GP Gaussian process. 11, 22, 24, 28

GPM Gaussian process model. 9, 10, 12, 13, 15, 21–26, 31, 37, 39, 40, 45, 61, 63–65, 96, 113, 114

GPS Gaussian process simulation. 11, 23, 25, 27, 28, 31, 34, 38, 39, 61, 100, 106, 113, 116

HMI Human-Machine Interaction. 90

I4.0 Industry 4.0. 8, 74, 75, 77

ICS Industrial Control Systems. 76

IGD Inverted Generational Distance. 104

IGD+ Inverted Generational Distance Plus. 104

IIC Industry IoT Consortium. 75

IIoT Industrial Internet of Things. 75, 76

IIRA Industrial Internet Reference Architecture. 14, 73, 75, 76, 80

IM Injection Molding. 29, 31

ISA Instance Space Analysis. 28

LHD Latin Hypercube Design. 31, 34, 43, 50, 52

MES Manufacturing Execution Systems. 77

ML Machine Learning. 9

MLE Maximum Likelihood Estimate. 22, 24, 25

MOEA Multi-objective Evolutionary Algorithm. 104, 105

MOEA/D Multiobjective Evolutionary Algorithm based on Decomposition. 105, 106

NFL No-free Lunch Theorem. 16

NSGA-III Nondominated Sorting Genetic Algorithm III. 105

OPCUA Open Platform Communications Unified Architecture. 57, 64, 69, 70, 97, 98, 113

PSO Particle Swarm Optimization. 44

PV Parameter Variation. 23, 26, 27, 31, 34, 36, 38, 39

RAMI4.0 Reference Architecture Model Industrie 4.0. 73, 75, 80

ROI Region of Interest. 18

SA Simulated Annealing. 44

SMBO Surrogate Model-Based Optimization. 14, 44, 63–65, 96, 98

SME Small and Medium-sized Enterprise. 10, 11

SMS-EMOA S-Metric Selection Evolutionary Multiobjective Optimization Algorithm. 105

SPOT Sequential Parameter Optimization Toolbox. 51, 65

t-SNE t-distributed Stochastic Neighbor Embedding. 33

VPS Versatile Production System. 69, 70, 95–99, 103, 107, 110, 117, 118

Bibliography

[1] Adolphs, P., Bedenbender, H., Dirzus, D., Ehlich, M., Epple, U., Hankel, M., Heidel, R., Hoffmeister, M., Huhle, H., Kärcher, B., Koziolek, H., Pichler, R., Pollmeier, S., Schewe, F., Walter, A., Waser, B., Wollschlaeger, M.: Reference Architecture Model Industrie 4.0 (RAMI4.0). Status report, VDI/VDE/ZVEI (2015). URL `https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2016/januar/GMA_Status_Report__Reference_Archtitecture_Model_Industrie_4.0__RAMI_4.0_/GMA-Status-Report-RAMI-40-July-2015.pdf`. Accessed: 2022-12-22

[2] Anders, D., Baum, M., Alken, J.: A comparative study of numerical simulation strategies in injection molding. In: 14th WCCM-ECCOMAS Congress 2020, pp. 1–11 (2021)

[3] Anderson, J.R.: A Simple Theory of Complex Cognition. American Psychologist **51**(4) (1996). DOI 10.1037/0003-066X.51.4.355

[4] Arora, S., Hu, W., Kothari, P.K.: An analysis of the t-sne algorithm for data visualization. In: Proceedings of the 31st Conference On Learning Theory, pp. 1455–1462 (2018)

[5] Atkinson, A.C., Fedorov, V.V., Herzberg, A.M., Zhang, R.: Elemental information matrices and optimal experimental design for generalized regression models. Journal of Statistical Planning and Inference **144**, 81 – 91 (2014). DOI 10.1016/j.jspi.2012.09.012

[6] Bäck, T., Fogel, D.B., Michalewicz, Z.: Handbook of Evolutionary Computation, 1st edn. IOP Publishing Ltd., GBR (1997)

[7] Barnard Feeney, A., Frechette, S., Srinivasan, V.: Cyber-physical systems engineering for manufacturing. In: S. Jeschke, C. Brecher, H. Song, D.B. Rawat (eds.) Industrial Internet of Things: Cybermanufacturing Systems, pp. 81–110. Springer International Publishing, Cham (2017). DOI 10.1007/978-3-319-42559-7_4

[8] Bartz-Beielstein, T., Doerr, C., van den Berg, D., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., Cava, W.L., Lopez-Ibanez, M., Malan, K.M., Moore, J.H., Naujoks, B., Orzechowski, P., Volz, V., Wagner, M., Weise, T.: Benchmarking in optimization: Best practice and open issues (version 2) (2020). ArXiv e-prints: 2007.03488v2

[9] Bartz-Beielstein, T., Doerr, C., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., Lopez-Ibanez, M., Malan, K.M., Moore, J.H., Naujoks, B., Orzechowski, P., Volz, V., Wagner, M., Weise, T.: Benchmarking in optimization: Best practice and open issues (version 1) (2020). ArXiv e-prints: 2007.03488v1

[10] Bartz-Beielstein, T., Preuss, M.: Automatic and interactive tuning of algorithms. In: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, p. 1361–1380. Association for Computing Machinery, New York, NY, USA (2011). DOI 10.1145/2001858.2002141

[11] Bartz-Beielstein, T., Rehbach, F., Rebolledo, M.: Tuning Algorithms for Stochastic Black-Box Optimization: State of the Art and Future Perspectives, pp. 67–108. Springer International Publishing, Cham (2021). DOI 10.1007/978-3-030-66515-9_3

[12] Bartz-Beielstein, T., Zaefferer, M., Rehbach, F.: In a nutshell – the sequential parameter optimization toolbox (2021). ArXiv e-prints: 1712.04076

[13] Bass, L., Clements, P., Kazman, R.: Software architecture in practice, 3 edn. Addison-Wesley (2012)

[14] Bergstra, J., Bengio, Y.: Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research (JMLR) **13**, 281 – 305 (2012)

[15] Beume, N., Naujoks, B., Emmerich, M.: Sms-emoa: Multiobjective selection based on dominated hypervolume. European Journal of Operational Research **181**(3), 1653–1669 (2007). DOI https://doi.org/10.1016/j.ejor.2006.08.008. URL `https://www.sciencedirect.com/science/article/pii/S0377221706005443`

[16] Bhatti, M.A.: Optimization Problem Formulation, pp. 1–45. Springer New York, New York, NY (2000). DOI 10.1007/978-1-4612-0501-2_1

[17] Bitkom, VDMA, ZVEI: Implementation Strategy Industrie 4.0. Report, Plattform Industrie 4.0 (2016). URL `https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2016/januar/Implementation_Strategy_Industrie_4.0_-_Report_on_the_results_of_Industrie_4.0_Platform/Implementation-Strategy-Industrie-40-ENG.pdf`. Accessed: 2022-12-22

[18] Blank, J., Deb, K.: pymoo: Multi-objective optimization in python. IEEE Access **8**, 89497–89509 (2020)

[19] Blank, J., Deb, K., Dhebar, Y., Bandaru, S., Seada, H.: Generating well-spaced points on a unit simplex for evolutionary many-objective optimization. IEEE Transactions on Evolutionary Computation **25**(1), 48–60 (2021). DOI 10.1109/TEVC.2020.2992387

[20] Bossek, J., Doerr, C., Kerschke, P., Neumann, A., Neumann, F.: Evolving sampling strategies for one-shot optimization tasks. In: T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, H. Trautmann (eds.) Parallel Problem Solving from Nature – PPSN XVI, pp. 111–124. Springer International Publishing, Cham (2020)

[21] Bousquet, O., Gelly, S., Kurach, K., Teytaud, O., Vincent, D.: Critical Hyper-Parameters: No Random, No Cry (2017). ArXiv e-prints: 1706.03200

[22] Breiman, L.: Random Forests. Machine Learning **45**(1), 5–32 (2001)

[23] Bunte, A., Fischbach, A., Strohschein, J., Bartz-Beielstein, T., Faeskorn-Woyke, H., Niggemann, O.: Evaluation of cognitive architectures for cyber-physical production systems. In: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 729–736 (2019). DOI 10.1109/ETFA.2019.8869038

[24] Bunte, A., Stein, B., Niggemann, O.: Model-based diagnosis for cyber-physical production systems based on machine learning and residual-based diagnosis models. In: Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19), pp. 2727–2735. Hawaii, USA (2019)

[25] Bunte, A., Wunderlich, P., Moriz, N., Li, P., Mankowski, A., Rogalla, A., Niggemann, O.: Why symbolic ai is a key technology for self-adaption in the context of cpps. In: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1701–1704. Zaragoza, Spain (2019)

[26] Bussemaker, J.H., Bartoli, N., Lefebvre, T., Ciampa, P.D., Nagel, B.: Effectiveness of surrogate-based optimization algorithms for system architecture optimization. In: AIAA AVIATION 2021 FORUM, p. 3095 (2021)

[27] Cauwet, M.L., Couprie, C., Dehos, J., Luc, P., Rapin, J., Riviere, M., Teytaud, F., Teytaud, O., Usunier, N.: Fully parallel hyperparameter search: Reshaped space-filling. In: H.D. III, A. Singh (eds.) Proceedings of the 37th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 119, pp. 1338–1348. PMLR (2020)

[28] CEN-CENELEC-ETSI Smart Grid Coordination Group: Smart Grid Information Security. Report (2012). URL `https://www.cencenelec.eu/media/CEN-CENELEC/AreasOfWork/CEN-CENELEC_Topics/Smart%20Grids%20and%20Meters/Smart%20Grids/security_smartgrids.pdf`. Accessed: 2022-12-22

[29] Chiarandini, M., Goegebeur, Y.: Mixed Models for the Analysis of Optimization Algorithms. In: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 225–264. Springer, Germany (2010). DOI 10.1007/978-3-642-02538-9

[30] Coello Coello, C.A., Reyes Sierra, M.: A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In: R. Monroy, G. Arroyo-Figueroa, L.E. Sucar, H. Sossa (eds.) MICAI 2004: Advances in Artificial Intelligence, pp. 688–697. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[31] Confluent: Schema management (2020). URL `https://docs.confluent.io/current/schema-registry/index.html`. Accessed: 2022-12-23

[32] Conover, W.J.: Practical nonparametric statistics, 3 edn. Wiley, New York, NY (1999)

[33] Conover, W.J., Iman, R.L.: On multiple-comparisons procedures. Tech. rep., Los Alamos Scientific Laboratory (1979). URL `http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-07677-MS`. Accessed: 2023-03-17

[34] Cressie, N.: Statistics for spatial data. John Wiley & Sons (2015)

[35] Das, I., Dennis, J.E.: A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. Structural optimization **14**(1), 63–69 (1997). DOI 10.1007/BF01197559. URL `https://doi.org/10.1007/BF01197559`

[36] Das, I., Dennis, J.E.: Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. SIAM Journal on Optimization **8**(3), 631–657 (1998). DOI 10.1137/S1052623496307510. URL `https://doi.org/10.1137/S1052623496307510`

[37] De Bie, T., De Raedt, L., Hernández-Orallo, J., Hoos, H.H., Smyth, P., Williams, C.K.I.: Automating data science. Commun. ACM **65**(3), 76–87 (2022). DOI 10.1145/3495256

[38] De Mauro, A., Greco, M., Grimaldi, M.: A formal definition of big data based on its essential features. Library Review **65**(3), 122–135 (2016). DOI 10.1108/LR-06-2015-0061. URL `https://doi.org/10.1108/LR-06-2015-0061`

[39] Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. IEEE Transactions on Evolutionary Computation **18**(4), 577–601 (2014). DOI 10.1109/TEVC.2013.2281535

[40] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. The Journal of Machine learning research **7**, 1–30 (2006)

[41] Devroye, L.: The compound random search. Ph.D. dissertation, Purdue Univ., West Lafayette, IN (1972)

[42] Dick, J., Pillichshammer, F.: Digital Nets and Sequences: Discrepancy Theory and Quasi–Monte Carlo Integration. Cambridge University Press (2010)

[43] Dietrich, K., Mersmann, O.: Increasing the diversity of benchmark function sets through affine recombination. In: G. Rudolph, A.V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, T. Tušar (eds.) Parallel Problem Solving from Nature – PPSN XVII, pp. 590–602. Springer International Publishing, Cham (2022)

[44] Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE computational intelligence magazine **1**(4), 28–39 (2006)

[45] Drath, R., Horch, A.: Industrie 4.0: Hit or hype? [industry forum]. IEEE Industrial Electronics Magazine **8**(2), 56–58 (2014). DOI 10.1109/MIE.2014.2312079

[46] Ehrgott, M.: Multicriteria Optimization, 2 edn. Springer (2005)

[47] Eiben, Á.E., Smit, S.K.: Evolutionary Algorithm Parameters and Methods to Tune Them. In: Autonomous search, pp. 15 – 36. Springer (2011)

[48] Eiben, Á.E., Smith, J.E.: Introduction to Evolutionary Computing, 2 edn. Natural Computing. Springer (2015)

[49] Elo, A.: The rating of chessplayers, past and present. Arco Pub, New York, NY, USA (1978)

[50] Emmerich, M.T.M., Deutz, A.H.: A tutorial on multiobjective optimization: fundamentals and evolutionary methods. Natural Computing **17**(3), 585–609 (2018). DOI 10.1007/s11047-018-9685-y. URL `https://doi.org/10.1007/s11047-018-9685-y`

[51] Fischbach, A., Bartz-Beielstein, T.: Improving the reliability of test functions generators. Applied Soft Computing **92**, 106315 (2020). DOI 10.1016/j.asoc.2020.106315

[52] Fischbach, A., Rehbach, F., Anders, D.: Data-driven problem classification and algorithm selection for injection molding optimization. Journal of International Scientific Publications: Materials, Methods & Technologies **16**, 59–74 (2022). URL `https://www.scientific-publications.net/en/article/1002508/`. Accessed 2022-12-23

[53] Fischbach, A., Strohschein, J., Bunte, A., Stork, J., Faeskorn-Woyke, H., Moriz, N., Bartz-Beielstein, T.: CAAI—a cognitive architecture to introduce artificial intelligence in cyber-physical production systems. The International Journal of Advanced Manufacturing Technology **111**(1), 609–626 (2020). DOI 10.1007/s00170-020-06094-z

[54] Fischbach, A., Zaefferer, M., Stork, J., Friese, M., Bartz-Beielstein, T.: From real world data to test functions. In: F. Hoffmann, E. Hüllermeier, R. Mikut (eds.) 26. Workshop Computational Intelligence, Proceedings, pp. 159–177. KIT Scientific Publishing, Dortmund (2016)

[55] Fletcher, R.: Conjugate gradient methods for indefinite systems. In: Numerical analysis, pp. 73–89. Springer (1976)

[56] Forrester, A., Sóbester, A., Keane, A.: Engineering design via surrogate modelling: a practical guide. John Wiley & Sons (2008)

[57] Gallagher, M., Yuan, B.: A general-purpose tunable landscape generator. IEEE Transactions on Evolutionary Computation **10**(5), 590–603 (2006)

[58] Gorecky, D., Schmitt, M., Loskyll, M., Zühlke, D.: Human-machine-interaction in the industry 4.0 era. In: 2014 12th IEEE International Conference on Industrial Informatics (INDIN), pp. 289–294 (2014). DOI 10.1109/INDIN.2014.6945523

[59] Grüner, S., Pfrommer, J., Palm, F.: Restful industrial communication with opc ua. IEEE Transactions on Industrial Informatics **12**(5), 1832–1841 (2016)

[60] Haftka, R.T.: Requirements for papers focusing on new or improved global optimization algorithms. Structural and Multidisciplinary Optimization **54**(1), 1–1 (2016). DOI 10.1007/s00158-016-1491-5

[61] Hancock, P., Kim, J.W.: Cognitive modeling of performance response capacity under time pressure. Tech. Rep. 2010-0907, Department of Psychology, University of Central Florida (2010). URL `https://apps.dtic.mil/sti/pdfs/ADA547369.pdf`. Accessed: 2022-12-23

[62] Hansen, N., Auger, A., Brockhoff, D., Tusar, D., Tusar, T.: COCO: performance assessment (2016). ArXiv e-prints: 1605.03560

[63] Hansen, N., Auger, A., Mersmann, O., Tusar, T., Brockhoff, D.: COCO: A platform for comparing continuous optimizers in a black-box setting (2016). ArXiv e-prints: 1603.08785v3

[64] Hansen, N., Finck, S., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Tech. Rep. RR-6829, INRIA (2009)

[65] Hightower, K., Burns, B., Beda, J.: Kubernetes: Up and Running Dive into the Future of Infrastructure, 1st edn. O'Reilly Media, Inc. (2017)

[66] Husslage, B.G., Rennen, G., Van Dam, E.R., Den Hertog, D.: Space-filling latin hypercube designs for computer experiments. Optimization and Engineering **12**(4), 611–630 (2011)

[67] Hutter, F., Kotthoff, L., Vanschoren, J.: Automated Machine Learning: Methods, Systems, Challenges. Springer (2019)

[68] IBM Knowledge Center: Main features and benefits of message queuing (2019). URL `https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q002630_.htm`. Accessed: 2022-12-23

[69] Ishibuchi, H., Masuda, H., Tanigaki, Y., Nojima, Y.: Modified distance calculation in generational distance and inverted generational distance. In: A. Gaspar-Cunha, C. Henggeler Antunes, C.C. Coello (eds.) Evolutionary Multi-Criterion Optimization, pp. 110–125. Springer International Publishing, Cham (2015)

[70] Jain, H., Deb, K.: An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: Handling constraints and extending to an adaptive approach. IEEE Transactions on Evolutionary Computation **18**(4), 602–622 (2014). DOI 10.1109/TEVC.2013.2281534

[71] James, W., Stein, C.: Estimation with quadratic loss. In: Proc. of the 4th Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1: Contributions to the Theory of Statistics, pp. 361–379. University of California Press (1961)

[72] Jasjeet S. Sekhon Walter R. Mebane, J.: Genetic Optimization Using Derivatives: Theory and Application to Nonlinear Models. Political Analysis **7**, 187–210 (1998)

[73] Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft computing **9**(1), 3–12 (2005)

[74] Jodlbauer, H., Schagerl, M.: Reifegradmodell industrie 4.0 - ein vorgehensmodell zur identifikation von industrie 4.0 potentialen. In: H.C. Mayr, M. Pinzger (eds.) Informatik 2016, pp. 1473–1487. Gesellschaft für Informatik e.V., Bonn (2016)

[75] Johnson, D.S., McGeoch, L.A.: Experimental Analysis of Heuristics for the STSP, pp. 369–443. Springer US, Boston, MA (2007). DOI 10.1007/0-306-48213-4_9

[76] Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization **13**(4), 455–492 (1998)

[77] Journel, A.G., Huijbregts, C.J.: Mining Geostatistics. Academic Press (1978)

[78] Jung, C., Zaefferer, M., Bartz-Beielstein, T., Rudolph, G.: Metamodel-based optimization of hot rolling processes in the metal industry. The International Journal of Advanced Manufacturing Technology **90**(1), 421–435 (2017). DOI 10.1007/s00170-016-9386-6

[79] Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on, vol. 4, pp. 1942–1948. IEEE (1995)

[80] Kerschke, P., Trautmann, H.: Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. Evolutionary Computation **27**(1), 99–127 (2019). DOI 10.1162/evco_a_00236

[81] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science **220**(4598), 671 – 680 (1983)

[82] Krige, D.: A statistical approach to some basic mine valuation problems on the witwatersrand. Journal of the Chemical, Metallurgical and Mining Society of South Africa **52**(6), 119–139 (1951)

[83] Kruskal, W.H., Wallis, W.A.: Use of ranks in one-criterion variance analysis. Journal of the American statistical Association **47**(260), 583–621 (1952)

[84] Kuhn, M.: Building predictive models in r using the caret package. Journal of Statistical Software, Articles **28**(5), 1–26 (2008). DOI 10.18637/jss.v028.i05

[85] Škvorc, U., Eftimov, T., Korošec, P.: Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis. Applied Soft Computing **90**, 106–138 (2020). DOI 10.1016/j.asoc.2020.106138

[86] Laird, J., Mohan, S.: Learning fast and slow: Levels of learning in general autonomous intelligent agents. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, pp. 7983–7987 (2018)

[87] Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An architecture for general intelligence. Artif. Intell. **33**(1), 1–64 (1987). DOI 10.1016/0004-3702(87) 90050-6

[88] Lang, R.D., Engelbrecht, A.P.: An exploratory landscape analysis-based benchmark suite. Algorithms **14**(3) (2021). DOI 10.3390/a14030078

[89] Lee, E.A., Seshia, S.A.: Introduction to embedded systems: A cyber-physical systems approach, 2 edn. Mit Press (2016). URL `https://ptolemy.berkeley.edu/books/leeseshia/`

[90] Lee, J., Bagheri, B., Kao, H.A.: A cyber-physical systems architecture for industry 4.0-based manufacturing systems. Manufacturing Letters **3**, 18 – 23 (2015). DOI 10.1016/j.mfglet.2014.12.001

[91] Lee, J., Jin, C., Bagheri, B.: Cyber physical systems for predictive production systems. Production Engineering **11**(2), 155–165 (2017). DOI 10.1007/s11740-017-0729-4

[92] Lehman, J.F., Laird, J., Rosenbloom, P.S.: A gentle introduction to soar, an architecture for human cognition:2006 update (1996)

[93] Li, D., Fast-Berglund, Å., Paulin, D.: Current and future industry 4.0 capabilities for information and knowledge sharing. The International Journal of Advanced Manufacturing Technology **105**(9), 3951–3963 (2019). DOI 10.1007/s00170-019-03942-5

[94] Liao, T., Molina, D., Stützle, T.: Performance evaluation of automatically tuned continuous optimizers on different benchmark sets. Applied Soft Computing **27**, 490–503 (2015). DOI 10.1016/j.asoc.2014.11.006

[95] Lin, S.W., Simmon, E., Young, D., Miller, B., Durand, J., Bleakley, G., Chigani, A., Martin, R., Murphy, B., Crawford, M.: The Industrial Internet Reference Architecture. Tech. rep., Industry IoT Consortium (2022)

[96] Maier, A., Niggemann, O.: On the learning of timing behavior for anomaly detection in cyber-physical production systems. In: International Workshop on the Principles of Diagnosis (DX), pp. 217–224 (2015)

[97] Malakuti, S., Bock, J., Weser, M., Venet, P., Zimmermann, P., Wiegand, M., Grothoff, J., Wagner, C., Bayha, A.: Challenges in skill-based engineering of industrial automation systems*. In: 2018 23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), vol. 1, pp. 67–74 (2018). DOI 10.1109/ETFA.2018.8502635

[98] Matoušek, J.: Geometric Discrepancy, 2 edn. Springer, Berlin (2009)

[99] Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory landscape analysis. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11, p. 829–836. Association for Computing Machinery, New York, NY, USA (2011). DOI 10.1145/2001576.2001690

[100] Mersmann, O., Preuss, M., Trautmann, H.: Benchmarking evolutionary algorithms: Towards exploratory landscape analysis. In: Parallel Problem Solving from Nature, PPSN XI, pp. 73–82 (2010)

[101] Mersmann, O., Preuss, M., Trautmann, H., Bischl, B., Weihs, C.: Analyzing the bbob results by means of benchmarking concepts. Evol. Comput. **23**(1), 161–185 (2015). DOI 10.1162/EVCO{\textunderscore}a{\textunderscore}00134

[102] Meunier, L., Chevaleyre, Y., Rapin, J., Royer, C., Teytaud, O.: On averaging the best samples in evolutionary computation. In: Proc. of Parallel Problem Solving from Nature, pp. 661–674. Springer (2020)

[103] Meunier, L., Doerr, C., Rapin, J., Teytaud, O.: Variance reduction for better sampling in continuous domains (2020). ArXiv e-prints: 2004.11687

[104] Minsky, M.: A Framework For Representing Knowledge, pp. 1–25. De Gruyter (1979). DOI doi:10.1515/9783110858778-003

[105] Monostori, L.: Cyber-Physical Systems, pp. 1–8. Springer, Berlin, Heidelberg (2018). DOI 10.1007/978-3-642-35950-7_16790-1

[106] Montgomery, D.C.: Design and Analysis of Experiments, 5th edn. Wiley, New York NY (2001)

[107] Morris, M.D., Mitchell, T.J.: Exploratory designs for computational experiments. Journal of Statistical Planning and Inference **43**(3), 381 – 402 (1995). DOI https://doi.org/10.1016/0378-3758(94)00035-T

[108] Mullen, K., Ardia, D., Gil, D., Windover, D., Cline, J.: Deoptim: An r package for global optimization by differential evolution. Journal of Statistical Software, Articles **40**(6), 1–26 (2011). DOI 10.18637/jss.v040.i06

[109] Muñoz, M.A., Kirley, M.: Sampling effects on algorithm selection for continuous black-box optimization. Algorithms **14**(1) (2021). DOI 10.3390/a14010019

[110] Muñoz, M.A., Kirley, M., Halgamuge, S.K.: The algorithm selection problem on the continuous optimization domain. In: C. Moewes, A. Nürnberger (eds.) Computational Intelligence in Intelligent Data Analysis, pp. 75–89. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[111] Muñoz, M.A., Smith-Miles, K.: Generating new space-filling test instances for continuous black-box optimization. Evolutionary computation **28**(3), 379–404 (2020)

[112] Muñoz, M.A., Smith-Miles, K.A.: Performance analysis of continuous black-box optimization algorithms via footprints in instance space. Evolutionary Computation **25**(4), 529–554 (2017). DOI 10.1162/evco_a_00194

[113] Muñoz, M.A., Sun, Y., Kirley, M., Halgamuge, S.K.: Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. Information Sciences **317**, 224 – 245 (2015). DOI 10.1016/j.ins.2015.05.010

[114] Narkhede, N., Shapira, G., Palino, T.: Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale, 1st edn. O'Reilly Media, Inc. (2017)

[115] Negus, C.: Docker Containers, 2nd edn. Addison-Wesley Professional (2015)

[116] Neisser, U.: Cognitive psychology, 1 edn. Appleton-Century-Crofts (1967)

[117] Nelder, J.A., Mead, R.: A simplex method for function minimization. The computer journal **7**(4), 308–313 (1965)

[118] Neugebauer, R., Hippmann, S., Leis, M., Landherr, M.: Industrie 4.0 - from the perspective of applied research. Procedia CIRP **57**, 2–7 (2016). DOI https://doi.org/10.1016/j.procir.2016.11.002. URL `https://www.sciencedirect.com/science/article/pii/S2212827116311556`. Factories of the Future in the digital environment - Proceedings of the 49th CIRP Conference on Manufacturing Systems

[119] Nuxoll, A.M., Laird, J.E.: Extending cognitive architecture with episodic memory. In: Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2, AAAI'07, p. 1560–1565. AAAI Press (2007)

[120] Otto, J., Vogel-Heuser, B., Niggemann, O.: Optimizing modular and reconfigurable cyber-physical production systems by determining parameters automatically. In: 2016 IEEE 14th International Conference on Industrial Informatics (INDIN), pp. 1100–1105 (2016). DOI 10.1109/INDIN.2016.7819329

[121] Pessot, E., Zangiacomi, A., Battistella, C., Rocchi, V., Sala, A., Sacco, M.: What matters in implementing the factory of the future: Insights from a survey in european manufacturing regions. Journal of Manufacturing Technology Management **32**, 795–819 (2020). DOI 10.1108/JMTM-05-2019-0169

[122] Plattform Industrie 4.0: Relationships between I4.0 Components – Composite Components and Smart Production. Working paper, Federal Ministry for Economic Affairs and Energy (BMWi) (2018). URL

https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/hm-2018-relationship.pdf. Accessed: 2021-05-31

[123] Plattform Industrie 4.0: Technology Scenario 'Artificial Intelligence in Industrie 4.0'. Working paper, Federal Ministry for Economic Affairs and Energy (BMWi) (2019). URL https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/AI-in-Industrie4.0.pdf. Accessed: 2022-12-23

[124] Pohlert, T.: Pmcmrplus: Calculate pairwise multiple comparisons of mean rank sums extended (2020). URL https://CRAN.R-project.org/package=PMCMRplus. Accessed: 2023-02-11

[125] Powell, M.J.D.: A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation, pp. 51–67. Springer Netherlands, Dordrecht (1994). DOI 10.1007/978-94-015-8330-5_4

[126] Powell, M.J.D.: The bobyqa algorithm for bound constrained optimization without derivatives. Tech. Rep. DAMTP 2009/NA06 (2009)

[127] R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2017). URL https://www.R-project.org/. Accessed: 2021-05-31

[128] Rajkumar, R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical systems: The next computing revolution. In: Design Automation Conference, pp. 731–736 (2010). DOI 10.1145/1837274.1837461

[129] Rechenberg, I.: Evolutionsstrategie '94. frommann-holzboog (1994)

[130] Rehbach, F., Zaefferer, M., Fischbach, A., Rudolph, G., Bartz-Beielstein, T.: Benchmark-driven configuration of a parallel model-based optimization algorithm. IEEE Transactions on Evolutionary Computation **26**(6), 1365–1379 (2022). DOI 10.1109/TEVC.2022.3163843

[131] Renau, Q., Doerr, C., Dreo, J., Doerr, B.: Exploratory landscape analysis is strongly sensitive to the sampling strategy. In: International Conference on Parallel Problem Solving from Nature, pp. 139–153. Springer (2020)

[132] Renau, Q., Dréo, J., Doerr, C., Doerr, B.: Towards explainable exploratory landscape analysis: Extreme feature selection for classifying bbob functions. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), pp. 17–33. Springer (2021)

[133] Rice, J.R.: The algorithm selection problem. In: M. Rubinoff, M.C. Yovits (eds.) Advances in computers, vol. 15, pp. 65 – 118. Elsevier (1976). DOI 10.1016/S0065-2458(08)60520-3

[134] Satterthwaite, F.E.: An approximate distribution of estimates of variance components. Biometrics Bulletin **2**(6), 110–114 (1946). DOI 10.2307/3002019

[135] Schröder, C.: The Challenges of Industry 4.0 for Small and Medium-sized Enterprises. Friedrich-Ebert-Stiftung, Abteilung Wirtschafts- und Sozialpolitik, Bonn, Germany (2016). URL `http://library.fes.de/pdf-files/wiso/12683.pdf`. Accessed: 2022-12-23

[136] Schumer, M.A., Steiglitz, K.: Adaptive step size random search. IEEE Transactions on Automatic Control **13**, 270–276 (1968)

[137] Shanno, D.F.: Conditioning of quasi-newton methods for function minimization. Mathematics of computation **24**(111), 647–656 (1970)

[138] Shi, Y., Eberhart, R.: A Modified Particle Swarm Optimizer. In: Proc. of the 1998 IEEE International Conference on Evolutionary Computation, within the IEEE World Congress on Computational Intelligence, pp. 69–73. IEEE (1998)

[139] Simcon Kunststofftechnische Software GmbH: Simulation of fluid flow and structural analysis within thin walled three dimensional geometries (EP 1385103 A1, 2004)

[140] Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. Computers & Operations Research **45**, 12–24 (2014). DOI 10.1016/j.cor.2013.11.015

[141] Smith-Miles, K.A.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. **41**(1), 1–25 (2009). DOI 10.1145/1456650.1456656

[142] Socha, K., Dorigo, M.: Ant colony optimization for continuous domains. European journal of operational research **185**(3), 1155–1173 (2008)

[143] Stein, C.: Inadmissibility of the usual estimator for the mean of a multivariate normal distribution. In: Proc. of the 3rd Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1: Contributions to the Theory of Statistics, pp. 197–206. University of California Press (1956)

[144] Stork, J., Eiben, A.E., Bartz-Beielstein, T.: A new taxonomy of global optimization algorithms. Natural Computing **21**, 219–242 (2020). DOI 10.1007/s11047-020-09820-4

[145] Stork, J., Friese, M., Zaefferer, M., Bartz-Beielstein, T., Fischbach, A., Breiderhoff, B., Naujoks, B., Tušar, T.: Open issues in surrogate-assisted optimization. In: High-Performance Simulation-Based Optimization, pp. 225–244. Springer (2020)

[146] Strohschein, J., Fischbach, A., Bunte, A., Faeskorn-Woyke, H., Moriz, N., Bartz-Beielstein, T.: Cognitive capabilities for the CAAI in cyber-physical production systems. The International Journal of Advanced Manufacturing Technology **115**(11), 3513–3532 (2021). DOI 10.1007/s00170-021-07248-3

[147] Watts, K.: Microservices Architecture: Deep Exploration Of Microservices. CreateSpace Independent Publishing Platform, North Charleston, SC, USA (2015)

[148] Weise, T., Chiong, R., Lassig, J., Tang, K., Tsutsui, S., Chen, W., Michalewicz, Z., Yao, X.: Benchmarking optimization algorithms: An open source framework for the traveling salesman problem. IEEE Computational Intelligence Magazine **9**(3), 40–52 (2014). DOI 10.1109/MCI.2014.2326101

[149] Wolpert, D.H.: What Is Important About the No Free Lunch Theorems?, pp. 373–388. Springer International Publishing, Cham (2021). DOI 10.1007/978-3-030-66515-9_13

[150] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (1997). DOI 10.1109/4235.585893

[151] Xiang, Y., Gubian, S., Suomela, B., Hoeng, J.: Generalized Simulated Annealing for Global Optimization: The GenSA Package. The R Journal **5**(1), 13–28 (2013). DOI 10.32614/RJ-2013-002

[152] Xing, B., Xiao, Y., Qin, Q.H., Cui, H.: Quality assessment of resistance spot welding process based on dynamic resistance signal and random forest based. The International Journal of Advanced Manufacturing Technology **94**(1), 327–339 (2018). DOI 10.1007/s00170-017-0889-6

[153] Zaefferer, M.: Surrogate models for discrete optimization problems. phdthesis, Technische Universität Dortmund (2018). DOI http://dx.doi.org/10.17877/DE290R-19857

[154] Zaefferer, M.: COBBS: Continuous optimization benchmarks by simulation. `https://github.com/martinzaefferer/COBBS` (2020). Accessed: 2022-12-23

[155] Zaefferer, M., Fischbach, A., Naujoks, B., Bartz-Beielstein, T.: Simulation based test functions for optimization algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference 2017, p. 905–912. ACM, Berlin, Germany (2017). DOI 10.1145/3071178.3071190

[156] Zaefferer, M., Rehbach, F.: Continuous optimization benchmarks by simulation. In: T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, H. Trautmann (eds.) Parallel Problem Solving from Nature – PPSN XVI, pp. 273–286. Springer International Publishing, Cham (2020)

[157] Zhang, Q., Li, H.: Moea/d: A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on Evolutionary Computation **11**(6), 712–731 (2007). DOI 10.1109/TEVC.2007.892759

[158] Zhong, R.Y., Xu, X., Klotz, E., Newman, S.T.: Intelligent manufacturing in the context of industry 4.0: A review. Engineering **3**(5), 616–630 (2017). DOI https://doi.org/10.1016/J.ENG.2017.05.015. URL `https://www.sciencedirect.com/science/article/pii/S2095809917307130`

[159] Zhou, K., Liu, T., Zhou, L.: Industry 4.0: Towards future industrial opportunities and challenges. In: 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), pp. 2147–2152 (2015). DOI 10.1109/FSKD.2015.7382284

[160] Zimmermann, P., Axmann, E., Brandenbourger, B., Dorofeev, K., Mankowski, A., Zanini, P.: Skill-based engineering and control on field-device-level with opc ua. In: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1101–1108 (2019). DOI 10.1109/ETFA.2019.8869473