universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Exploring performance of active learning for optimization“

verfasst von / submitted by

Clemens Wager, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2024 / Vienna, 2024

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 910
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Computational Science (Master)
Betreut von / Supervisor:	Univ.-Prof. Dr. Norbert Mauser
Mitbetreut von / Co-Supervisor:	Dipl.-Ing. Dr.techn. Lukas Exl

Acknowledgements

The financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged. Further, I acknowledge the support of the University for Continuing Education Krems for providing me with a professional working environment and working materials.

I gratefully acknowledge the support and feedback of my supervisor Univ.-Prof. Dr. Norbert Mauser as well as my co-supervisor Dipl.-Ing. Dr.techn. Lukas Exl. Special thanks go to Univ.-Doz. Dipl.-Ing.Dr. Thomas Schrefl who was a great help to me with his advice, experience and ideas. Further, I want to thank Dr. Alexander Kovacs, MSc BSc for introducing me to the project group and sparking my interest in this fascinating subject. I am grateful for the patience he welcomed me with and for the expertise, motivation and support I received from him. Moreover, I want to thank my colleagues from the project group Ing. Dr. Harald Özelt, MSc, Claas Fillies, BSc, and many more for being a great team, supporting me with their knowledge, fun distractions and great experience. Finally, huge thanks go to my parents Mag.[a] Sonja Wager and Georg Wager, MBA, MLE, my brother Matthias Wager, BSc and my girlfriend Marie Diernhofer, BSc.

I look forward to working in this field and collaborating with the colleagues and friends I have met during this project. As I reflect on the future, I fondly remember the advice I received in various situations: "Clemens, mach keine Doktorarbeit daraus!" ("Clemens, don't turn it into a doctoral thesis!"), though I might just do that.

Abstract

Optimization problems are ubiquitous in various fields such as nature, economics, engineering, etc., presenting significant challenges in finding efficient solutions, i.e., for material design or shortest pathways. The complexity of these problems often lies in their multi-objective nature and the presence of noisy, high-dimensional landscapes. The optimization of those problems becomes remarkably tough if the evaluation of objective functions is computationally expensive leading the calculation processes to consume exceedingly high amounts of time and/or energy. Active learning schemes (ALS) have emerged as a promising approach to address these challenges. In general, an ALS operates by iteratively refining a machine learning surrogate model to approximate the objective function based on new data points proposed to explore, thereby enhancing the performance of the optimization process. This thesis examines the effectiveness and efficiency of an ALS in solving complex optimization tasks on single- and multi-objective problems.

A modular Python framework was developed to conduct experiments on an active learning scheme with various configurations. The genetic algorithm NSGA-II serves as a benchmark against which the performance of the ALS is compared. Both solving approaches were inspected in a comparative parameter study. This study was set up with different sample sizes and four different sampling methods on six optimization test functions.

The research finds that the ALS is easily about 4-10 times more effective than the genetic algorithm in simple environments. However, its limited ability to explore complex, noisy landscapes leads to suboptimal solutions. The results reveal that while the ALS finds plenty of Pareto optimal solutions in multi-objective functions, it only has limited capacity to adequately explore the whole Pareto set. Additionally, the findings suggest that different sampling methods only play a minor role in the overall efficiency of both solving approaches. Hence, to address these shortcomings, this thesis proposes several improvements, including the integration of uncertainty measures like standard deviation through Gaussian Process regressors as surrogate model and the application of mini-batching techniques in the model training process. Other refinements include, i.e., a mechanism to prevent the potential re-evaluation of data points close to already known ones. These enhancements aim to broaden the ALS's exploration capability and improve its performance in complex, resource-demanding optimization problems.

In summary, this thesis contributes to the ongoing discourse in optimization research by critically examining the capabilities of the active learning scheme in single- and multi-objective optimization and proposing potential pathways for its improvement. The findings

serve as a proof-of-concept, paving the way for more comprehensive and statistically significant future studies in this domain.

Contents

List of Tables

List of Figures

1. Introduction

Optimization is a ubiquitous concept. It is prevalent in our daily lives when we use GPS navigation systems to find the quickest route during rush hour traffic, in engineering when designing energy-efficient buildings, in the economy when companies optimize supply chains to minimize costs, and in nature when organisms evolve to maximize their chances of survival and reproduction, such as the efficient foraging behavior of animals. Both nature and humans have a natural tendency to find the best possible solution to a given problem [1].

There exists a multitude of algorithms to solve optimization problems computationally to discover the most efficient solutions. For some problems these search strategies use exceeding amounts of resources, especially if complex high-fidelity simulations or computationally expensive high-dimensional functions are involved [2]. Many engineering disciplines rely on such demanding optimization tasks [2], as well as the field of material design, where we optimize, i.e., material structure or chemical compositions to find improved materials with beneficial characteristics [3, 4]. However, solving such problems became increasingly relevant in recent years, i.e., to lower energy consumption in buildings [5, 6] or to optimize permanent magnets for electric engines [3, 7, 8, 9]. Hence, there is a growing need for advanced optimization techniques.

This work is motivated by the desire to study and explore a promising technique that leverages the potential of data-driven methods to solve such complex optimization tasks. The concept of active learning schemes (ALS), has been around for some time, but has gained considerable attention and utilization in recent research as an effective approach to optimization [3, 10, 11, 12]. These schemes offer the potential to enhance the way we address optimization problems by reducing resource requirements, saving time, and conserving energy [12]. The integration of data-driven methods, that have flourished in the past two decades [13, 14, 15, 7], in these schemes, presents an exciting opportunity to harness the power of data for optimization tasks. With their intelligent exploration, active learning schemes promise to tackle these challenges more efficiently [12].

In response to this need, this thesis has been dedicated to developing and studying an ALS tailored for optimization tasks. By delving into the capabilities of this ALS approach and laying the groundwork for future research, this work aims to illuminate the advantages and potential improvements that active learning schemes can bring to optimization. An extensive comparative parameter study has been conducted to empirically evaluate the performance of the ALS, providing a solid foundation for understanding its impact.

The structure of this work is carefully designed to guide the reader through the topics of optimization, machine learning, meta modelling, and the core subject of active learning schemes. Beginning with a foundational overview of global optimization, the document progresses to discuss the pivotal role of machine learning and introduces the concept of meta models, which play a central role in active learning schemes. The subsequent chapters delve deeper into the specifics of the ALS, examining its architecture, potential outcomes, and motivations for its use, alongside a critical evaluation of its limitations. Leading into the empirical body of this work, the subsequent chapter describes the detailed parameter study in which the experimental setup and the Python framework, we developed for this purpose, are explained. This study not only showcases the practical application of the ALS in optimization but it also sets the stage for a discussion on the results and implications in the final chapters.

This thesis provides a comprehensive exploration of the active learning scheme in optimization. The potential strategies outlined in this study offer a roadmap for future research and development in this field.

2. Global Optimization

This chapter introduces the topic of optimization and explains the fundamental background. The mathematical formulation of optimization problems and the concept of the Pareto frontier in the context of multi-objective optimization are presented. Then two representative optimization algorithms are discussed and finally the concepts of convergence and termination criteria are introduced.

2.1. Fundamentals

This introductory section draws extensively from the foundational work of Nocedal and Wright in their book *Numerical Optimization* from 2006 [1], where they provide valuable insights into the field of optimization. The concepts and ideas presented in this section are rooted in their research and contributions.

Optimization is a fundamental human behavior. Investors strive to create portfolios that balance risk and return, while manufacturers seek to maximize the efficiency of their production processes. Engineers carefully fine-tune parameters until the performance of their designs is optimized.

Nature is a master of optimization. Physical systems gravitate towards a state at which energy is minimal. In an isolated chemical system, molecules react with one another until their electrons' total potential energy reaches a minimum. Certain slime molds are able to find the shortest path through a maze minimizing their effort to traverse from entry to exit. In other words, both nature and humans have a natural tendency to find the best possible solution to a given problem. This is driven by the desire to minimize effort, maximize efficiency, and achieve the best possible outcome.

In order to apply the powerful tool of optimization to decision making processes or to the analysis of physical systems, we first need to identify an *objective*. It will measure our system's performance quantitatively. The objective depends on specific properties of the system, which we call *variables*. Every optimization aims at finding the best set of input variables that will optimize the objective. Some variables are subject to *constraints*, e.g., the velocity of a particle cannot become negative. These constraints must be satisfied by the optimization. The process of identifying the objective, the variable(s) and the constraint(s) is called *modelling*. It is the first and one of the most important steps during the optimization. There exist also systems with many objectives called *multi-objective* optimization problems. However we will focus on *single-objective* optimization problems

for now and cover the multi-objective case in detail in Section 2.3.

Finally, we solve the problem by applying an *optimization algorithm* (Section 2.4). There exist many algorithms with different strengths and limitations. The choice is crucial, because some algorithms may not be able to solve the problem efficiently. In the worst case, it may not have a solution at all. Therefore, it is essential to analyze the modeled optimization problem carefully to choose the right solving strategy [1].

2.2. Mathematical Formulation

The objective is mathematically formulated as the *objective function* $f(\cdot) : \mathbb{R}^{k_{\text{in}}} \mapsto \mathbb{R}^{k_{\text{out}}}$, where $k_{\text{in}}, k_{\text{out}} > 0$ represent the dimension of the input space $\mathbb{R}^{k_{\text{in}}}$ and the output space $\mathbb{R}^{k_{\text{out}}}$, respectively. Usually the input space is also called *decision space* or *design space*, while the output space is also referred to as *objective space*. Constraints can restrict x and $f(x)$ and they can be formulated as either equality constraint, i.e., $c(x) = 0$ or inequality constraint, i.e., $c(x) \geq 0$. The subset of the input space that satisfies all constraints is the *feasible region* $\mathbb{F}$. Thus, the solution to the (single-objective) optimization problem $x^* \in \mathbb{F}^{k_{\text{in}}}$, where x is a vector of dimension k_{in} [16].

When the objective function $f(x)$ is identified we can formulate, for instance, a constrained single-objective *optimization problem* as

$$\min_{x \in \mathbb{R}^{k_{\text{in}}}} f(x) \text{ subject to } \begin{matrix} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I}, \end{matrix} \tag{2.1}$$

where x is a vector of input variables, c_i are scalar functions of x that constrain the optimization and $\mathcal{E}$ and $\mathcal{I}$ are the finite sets of indices of equality and inequality constraints, respectively. By convention the formulation considers the minimization of the objective function, because the minimization of $f(\cdot)$ is equivalent to the maximization of $f(\cdot) := (-1)\ g(\cdot)$ [1].

2.3. Characteristics of Multi-Objective Optimization

A *multi-objective optimization* problem has $k_{\text{out}} > 1$ objective functions, which map the input vector x from the decision space to the k_{out}-dimensional objective space. The objective functions of multi-objective optimization problems may conflict with each other. If that is the case, no single optimal solution x^* is achievable, but rather a set of solutions which each satisfy the objective functions to different degrees. However, the objective functions do not have to conflict with each other. Then, there exists a single optimum x^*. In the conflicting case, the solutions can be categorized in Pareto efficient and non-efficient (Section 2.3.1) [17].

Properties of $F(x)$	Properties of $\{c_i(x)\}$
Function of a single variable	No constraints
Sum of squares of linear functions	Simple bounds
Quadratic function	Sparse linear functions
Sum of squares of nonlinear functions	
Linear functions	
Smooth nonlinear function	
Sparse nonlinear function	
Non-smooth nonlinear function	

Table 2.1.: Typical classification scheme based on the properties of the problem functions. This table is an adaptation of the table in the book "*Practical Optimization*" by Gill et al. [21].

2.3.1. Pareto Efficiency

Let us briefly dive into the concept of the *Pareto efficiency* or *Pareto optimality*. There exists a *Pareto set* $\mathcal{PS} \subset \mathbb{R}^{k_{\text{in}}}$ in the decision space, where each element is a feasible and a minimal solution to the objective problem. Analogously, in the objective space there exists a *Pareto front* $\mathcal{PF} \in \mathbb{R}^{k_{\text{out}}}$ which is the image of $\mathcal{PS}$ [17, 18].

It is important to distinguish between the *true* Pareto set $\mathcal{PS}^*$, the unique solution to a problem, and the *computed* Pareto set $\hat{\mathcal{PS}}$, an approximation produced by an optimization algorithm (Section 2.4). Comparing these two sets and their images allows us to measure algorithm convergence, which is explained in Section 2.4.4.

We can formulate the concept of the Pareto set as it being a "*family of points which is optimal in the sense that no improvement can be achieved in any objective without degradation in others*" [19]. In terms of the Pareto concept we say that a point x_0 *dominates* another point x_1, if x_0 is more efficient or optimal than x_1. All members of the Pareto set are optimal and are therefore *non-dominated* by any other points [20]. After the Pareto set was computed it is up to the decision maker to choose one of the Pareto optimal decision variables x that best satisfies the real world optimization problem. Figure 2.1 illustrates how a solution is considered more Pareto efficient than another.

2.4. Optimization Algorithms

We solve an optimization problem computationally. After analyzing the optimization task carefully an algorithmic approach to solve for the best decision variable x is chosen. The choice depends heavily on the problem functions' properties. For this reason, we characterize the problem's nature in order to facilitate this choice. In Table 2.1 a few typical function properties are listed. Certain algorithms are well suited for solving certain problem classes, because we can take algorithmic advantage of each of those characteristics.

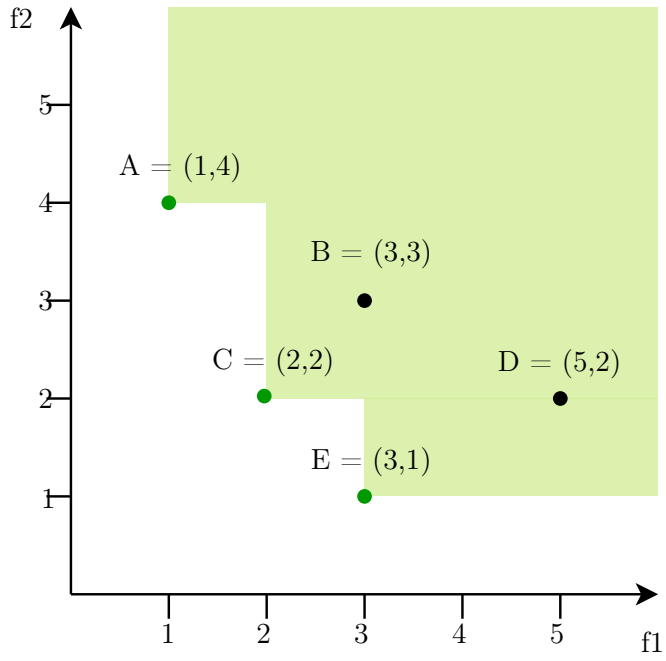


Figure 2.1.: Pareto Efficiency
There are five solutions (A, B, C, D and E) for a minimization problem with two objective functions $f1$ and $f2$. The two dimensions of the objective space are plotted against each other and the objective function value of each solution is plotted as well. The green points A, C and E are Pareto optimal which means they are non-dominated. These green solutions span a space marked in light green where all dominated points are located, i.e., B and D. So, there exist more Pareto efficient solutions than B and D that improve at least one objective function [20].

Also, the size of the problem meaning the number of variables has a huge impact on the choice of the solution approach. However, the "size" of a problem cannot be defined in absolute terms, as it is subject to trade-offs that depend on other problem properties. Further, available resources like time and memory on the computer need to be considered. The problem size greatly determines the amount of computer resources needed. Lastly, there may be some external factors that affect the algorithm choice. In some cases, it may be a waste of resources to compute highly accurate results if their accuracy is not relevant, e.g., if the result will be refined later or if a lower accuracy meets the requirements of our use case. Naturally, we pick the algorithm that will solve the problem most efficiently [21]. In the following two iterative strategies will be introduced that output estimates of the solution that hopefully gravitate towards the true solution x^*.

2.4.1. Gradient Descent Method

The gradient descent (GD) method is a popular technique to minimize the objective function $f(x)$ by updating the decision variables in the negative direction of the gradient of the objective function $\nabla_x f(x)$ w.r.t. the decision variables. The step size in each iteration is given by η which defines the distance the algorithm steps into the direction of the gradient.

We can visualize the objective function as a surface, and its gradient as the direction of that surface's slope. To find the minimum of the objective function, we follow the gradient downhill until we reach a valley, where the gradient is zero [22].

When the algorithm is allowed to choose the step length η it shows very good convergences behaviour. Then we also refer to η as an "adaptive step length". Typically, a *line search* algorithm is used to find the sufficiently small step length [22].

A slowly decreasing η in stochastic gradient descent (SGD), a variant of the GD method that estimates the gradient, has shown good convergence behaviour. Hence, gradient descent methods, i.e. SGD, reach a global minimum for convex objective functions, whereas for a non-convex functions only a local minimum may be reached [21, 22]. However, it is not guaranteed that the computed solution will be the global optimum [21].

It is important to note, that the gradient descent method can only be applied if the objective function is continuous and differentiable.

Gradient Descent Akin Method

The vanilla gradient descent method requires some adjustments to solve **constrained** optimization problems. For instance, Chen et al.[23] suggest a variant of the gradient descent method called "gradient descent akin method" (GDAM) where "*the search direction is computed using a linear combination of the negative and normalized objective and constraint gradient*".

2.4.2. Genetic Algorithms

Genetic algorithms are general-purpose optimization algorithms that sacrifice performance for robustness in finding a global solution. Basically, genetic algorithms search for optimal solutions by performing an artificial version of Darwinian evolution ("*the survival of the fittest*") [24]. The terminology of genetic algorithms is strongly based on that of evolutionary theory. A solution to the optimization problem is an *individual* and a group of individuals is a *population*, while a new population is called a *generation*. Each individual has a *fitness* that corresponds to the value of the objective function(s) evaluated on the individual (=solution) [20]. The algorithm uses a heuristic approach instead of following a deterministic set of rules. Therefore, three genetic operators following nature's example are used to control the evolution of successive generations:

1. The probability of ***reproduction*** of a particular individual is proportional to its fitness.
2. ***Crossover*** recombines very important features in parent chromosomes to allow the subsequent generation to inherit their best features and to approximate the optimal solution.
3. ***Mutation*** implies that some of an individual's inherent characteristics will be changed randomly to introduce background variation.

Certain variations of the genetic algorithm may use slightly different genetic operators, but these are the fundamental principles [20, 24].

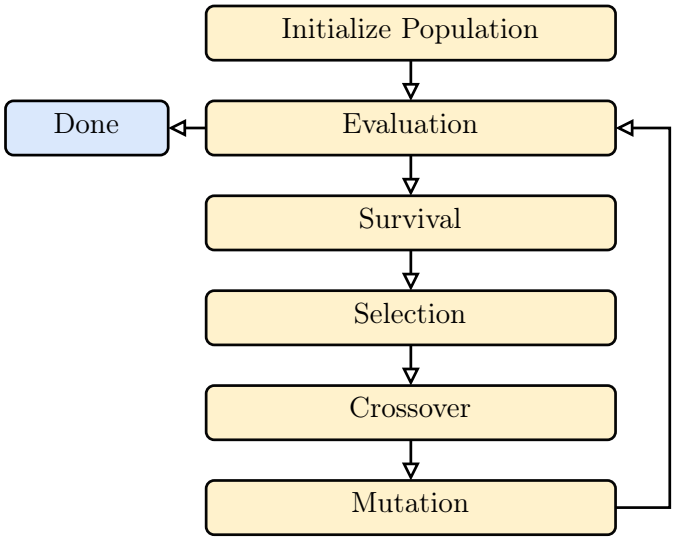


Figure 2.2.: Flow of genetic algorithm
The flowchart shows which stages the population of a genetic algorithm pass through. This figure is an adaptation from [25] and is also explained in the pymoo documentation [1].

Upon initialization of a genetic algorithm a new generation is spawned, with each individual being a possible solution to the optimization problem. Then fitness of the evaluation is evaluated and the genetic operators are applied. Thus the second generation is created and it inherits the best features of its parent generation. Therefore, its average fitness is higher than the previous generations. Their fitness is evaluated as well and the genetic operators produce the next subsequent generation. This cycle continues for as many iterations as the user has configured by implementing a stopping condition (explained in the next section) [20].

Genetic algorithms are a popular choice for solving optimization problems and especially for the multi-objective case, because they can find multiple Pareto optimal solutions

in one run. The genetic algorithms belong to the general-purpose algorithms, meaning that they can be applied to a wide range of optimization problems without requiring much specific knowledge about the problem. This makes them a good first choice for solving new problems. They do not even require the objective function to be continuous. Further, genetic algorithms can effectively handle high-dimensional optimization problems. Additionally, Genetic algorithms are able to escape local minima by using stochastic operators such as crossover and mutation. This makes them less likely to get stuck in poor solutions [26, 20].

Some popular examples for optimization algorithms used in practical application include Multi-Objective Genetic Algorithm(MOGA) [5], Non-dominated Sorting Genetic Algorithm 2 (NSGA-II) [6], Pareto Genetic Algorithm (ParetoGA) [27], Multi-Island Genetic Algorithm (MIGA) [28] and (RCGA) [29].

Comparison to GD Method

When we compare these population-based optimization algorithms with the gradient descent methods we can notice a few profound differences which heavily influenced the algorithm choice in this thesis.

First, genetic algorithms have a fairly robust convergence behaviour and can solve non-differentiable and discontinuous objective functions, while the gradient descent methods cannot [24].

Second, the genetic optimization algorithm is able to find not just a single solution for a multi-objective problem, but it generates a whole population where each individual is a Pareto efficient solution. Plotting the function values of this Pareto set $\mathcal{PS}$ in the objective space lets us visualize the Pareto front $\mathcal{PF}$. The information in $\mathcal{PF}$ illustrates the trade-offs in the problem's domain, e.g. trade-offs in price and quality of a product and can thus helps us in decision making. Hence, this solution set provides a more complete understanding of to problem than just a single solution, which a gradient descent method would find [20].

Lastly, while constraint handling is possible in some GD methods (i.e. GD Akin method in Section 2.4.1) a much more comprehensible way of constraint handling is possible for the population of a genetic algorithm. Basically, when new individuals are created, each one can be analysed to determine whether it violates violates constraints or not. The individuals that violate a restriction can then excluded from the population or their characteristics can be changed in order to "repair" them. The latter approach is implemented in pymoo [25] in the *Repair* module [2].

[2] https://pymoo.org/operators/repair.html

2.4.3. Global Convergence

Convergence in algorithms is the property where the sequence of values generated by the algorithm approaches a specific limit L as the number of iterations increases, with the difference between the generated values and the limit becoming arbitrarily small. This property is defined mathematically as follows:

Definition 2.4.3 (Global Convergence). An algorithm converges globally if, for a given tolerance $\epsilon > 0$, there exists an iteration N such that for all $n > N$, $|x_n - L| < \epsilon$, where x_n is the design variable with a certain value at iteration n and L is the limit to which it converges.

2.4.4. Convergence metrics

When we quantify the convergence of **single-objective** optimization algorithms and we know the exact position and value of the global optimum we can compute the *Euclidean distance* d in the objective space between the algorithm's solution a and the true solution a^* given by

$$d(a, a^*) = \sqrt{(a - a^*)^2} \quad , \tag{2.2}$$

where $a, a^* \in \mathbb{R}^1$.

In the case of **multi-objective** optimization there are several metrics that consider different aspects of the solution set, such as capacity, convergence, diversity or convergence-diversity. A few well established measures for these aspects include hypervolume (HV), generational distance (GD), inverted generational distance (IGD), Averaged Hausdorff distance (p), Spread(Δ) and R2-indicator [18]. The properties of the computed solution can be analyzed in the decision space and in the objective space. However, it is most useful to assert the quality of the solution in the objective space, because there we can measure the objective function value $f(x)$ which we want to be minimal. So, typically the computed Pareto frontier is analyzed. But it depends on the information about the Pareto optimal solution we have. In some cases we may have to rely on the information we have in the decision space.

We can measure an algorithm's convergence towards a true or approximated Pareto front using the metric of *Generational Distance (GD)* or *Inverse Generational Distance (IGD)*. For this purpose, a reference set of points on the Pareto front is required.

Let $Z = \{z_1, z_2, ..., z_n\}$ be such a set of n reference points on the Pareto front in the objective space $\mathbb{R}^{k_{\text{out}}}$. We want to measure the convergence of a computed solution set $A = \{a_1, a_2, ..., a_m\}$, holding m points where each point $(a_1, a_2, ..., a_m)$ was produced by an optimization algorithm and also lies in the objective space $\mathbb{R}^{k_{\text{out}}}$. Then the IGD is given by

$$IGD(Z, A) = \frac{1}{n} \sum_{j=1}^{n} \min_{a_i \in A} d(a_i, z_j) \quad , \tag{2.3}$$

where $d(a_i, z_j)$ is the Euclidean distance (Equation 2.2) between a_i and z_j. Here, i is the index of the computed solutions in A, where a_i is the point closest to the current point z_j on the Pareto front. So, the IGD is the average Euclidean distance between each reference point in Z and the next closest point in A. The concept of the IGD is visualized in Figure 2.3, where we see the red reference points and how each one of them is associated with the next solution point colored in green.

For the sake of completeness, the definition of the Generational Distance (GD) is similar, but it first considers each point a_i and computes the Euclidean distance to the next closest point z_j. Thus, the GD may not consider all reference points in Z, which may not be a suitable property in some cases [30, 18].

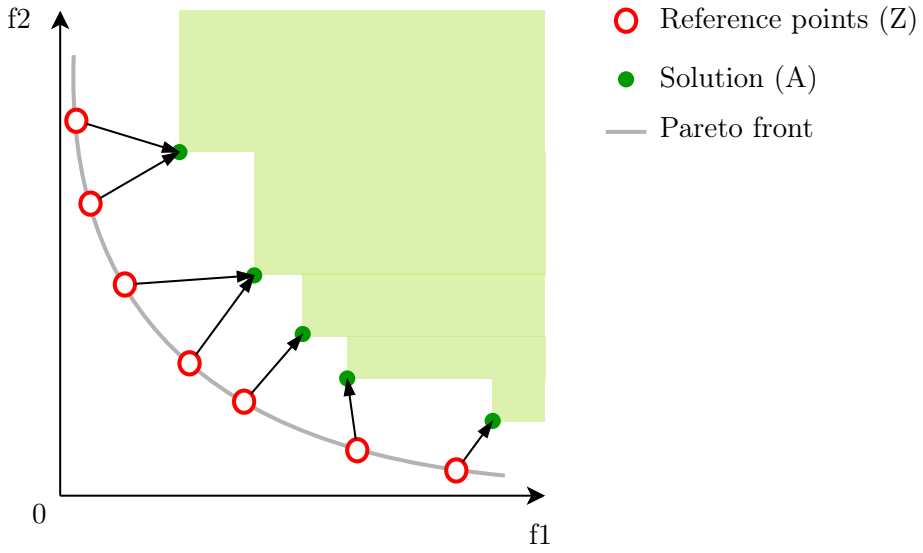


Figure 2.3.: Inverse Generational Distance (IGD)
This figure shows a plot of two objective functions $f1$ and $f2$ plotted against each other with a Pareto front in grey, reference points of Z on the front in red and solutions of A in green. The arrows illustrate the Euclidean distance between each reference point and its next solution. IGD is defined as the average of the sum of these distances. This figure was inspired by an image in *Evolutionary Multi-Criterion Optimization* by Deb et al. [30].

2.5. Termination Criteria

Termination criteria are essential for iterative optimization algorithms to indicate when to break a loop that, i.e., that repeatedly performs an operation, or to stop optimizing the objective function. The goal is to terminate the algorithm when it *converged*, meaning we obtained the global or a local optimum; or at least when we found a solution that is good enough. So, there has to be at least one predefined criterion or a set of criteria that will stop the search if one or many conditions are met. However, since the optimal solution is unknown at the beginning of the search, we need to define a case that occurs immediately before or after convergence.

The first and simplest solution is to set a hard limit for the number of iterations. One of the main motivations for this approach can be that there are only limited computational resources and letting the iteration run for longer will become too expensive. Second, there can be some control mechanism to monitor whether the algorithm's exploration slows down or even gets to a halt. Analogously, we can check if the computed solution does not improve anymore by a prescribed threshold. In multi-objective space, one way would be to investigate if the Pareto front advances or changes in shape. When the algorithm terminates due to some set limit, we might end up with a minimum, which can be either global in the best case or it is just a local minimum that needs further investigation.

Lastly, if the optimal solution or solution set is known, for example, if we want to benchmark an algorithm and test it on a well explored test function, we can stop the iteration when the solution is within a certain tolerance of the known minimum or the known Pareto front. For this purpose we can use the Euclidean distance (Equation 2.2) in single-objective optimization determine the distance between the computed solution $\hat{x}$ and the optimum x^* or in the multi-objective case the IGD (Equation 2.3) to check the approximation of the computed towards the true Pareto front.

3. Machine Learning

This chapter sets the stage for machine learning due to its key role in the active learning scheme. Some domain specific terminology and the basics of the machine learning approach are presented. This encompasses the introduction of learning algorithms, models and also typical drawbacks of machine learning are discussed.

3.1. Algorithms

A machine learning algorithm processes a dataset and learns from it. When we apply such a *learning* algorithm in a computer program we typically observe that the program becomes better at solving its task the more it interacts with the data and the more data it receives.

From an engineer's perspective machine learning is mainly interesting due to its ability to solve complicated tasks that are too difficult to tackle with a static code written by humans which explicitly says what must be done line by line. Studying this learning process is fascinating, because from a philosophical point of view it allows us to observe one of the basic principles of intelligent behaviour. When we employ such a learning algorithm, its task is only to solve a certain problem without being given explicit directions on *how* to accomplish the task. The program must find the *how* (meaning a solving strategy) on its own. There exist many different types of machine learning algorithms for different challenges, including problems such as *classification*, *transcription*, *translation*, *anomaly detection* and *regression* [14].

3.1.1. Implementation

When working with machine learning algorithms computer scientists typically use the programming language *Python*. There is a multitude of software packages available online that implement various machine learning techniques, such as scikit-learn [31] and TensorFlow [32].

In practice, if we want to create a machine learning model, we would write a computer program that uses a machine learning algorithm appropriate for the task and the available data, and instruct it, e.g., to predict a numerical value for a given input. The program will train the algorithm and use the model created to make a prediction. We can also save the model and reuse it later [14].

3.1.2. Training

Machine learning algorithms "interact" or "experience" a dataset in order to learn from it. Generally, this interaction process, called "*training*", happens once during the development of a machine learning model (Section 3.2). After the algorithm has processed all the provided data, the training is complete and the stored knowledge can be applied to a task [14]. This training phase can be considered a "passive" process, because the algorithm takes no "active" part in the development of the model, when we compare the machine learning approach to the *active learning* approach (Chapter 5). The algorithm is merely applied to process the data it is provided with [12].

3.1.3. Supervised and Unsupervised Learning

Generally, there are two big subgroups of machine learning algorithms, namely *supervised learning* and *unsupervised learning*, however there is no formal definition.
Supervised learning algorithms are trained on *labeled* datasets, where each example consists of an input and an associated desired output (called "*label*"). Once trained, supervised learning algorithms can be used to predict the label for *unseen* (= new) examples.
Unsupervised learning algorithms are trained on unlabeled datasets, where the output for each example is unknown. These algorithms learn the underlying structure and (hidden) patterns within the data. They can be used to cluster data, identify patterns, and detect anomalies.

It is worth noting, that there are also other types of learning, i.e., *semi-supervised learning* and *reinforcement learning*, which however do not matter in the scope of this thesis [14].

3.1.4. Neural Networks

The concept of neural networks was developed in the 1980s leaning on findings from neuroscience. Essentially, the idea is to let many highly interconnected computational units (called "*neurons*") interact with each other in an *artificial* neural network (ANN), just like neurons interact in a brain. The Neurons are the information processing units of a neural network and when they are arranged on the same level within the network they form a so-called *layer*. An artificial neural network is made up of several successive layers which is illustrated in Figure 3.1.

Each neuron possesses a weight that regulates the strength of the connection to other neurons and a non-linear function (referred to as *activation function*) that the input passes through to produce an output. While the neural network is being trained, the weights adjust to optimize the network's performance [14, 13]. Each neuron takes an array of real-valued inputs and produces one real-valued output using its activation function. This output may become a part of the input array of neurons in the following layer [13].

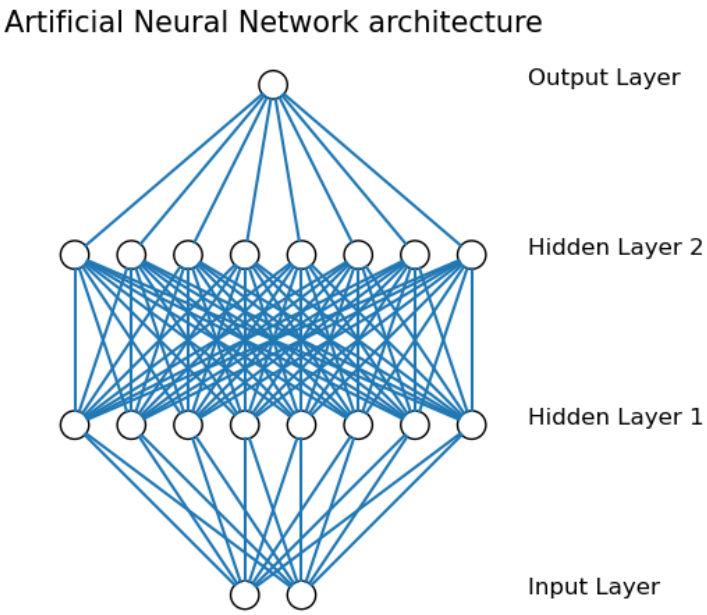


Figure 3.1.: Visualization of an artificial neural network's architecture with the following structure: [2, 8, 8, 1]. This ANN is build from one input layer with 2 neurons, two hidden layers with 8 neurons each and an output layer with 1 neuron. Layers next to each other are densely connected.

Neural networks can learn complex non-linear functions through supervised learning [33].

Typically, each layer has a certain purpose, e.g. the input layer is the initial contact point of the network with the provided data, the output layer returns the desired result vector and hidden layers in between can serve many different roles. Hence, we describe neural networks on a surface level by means of the resulting *architecture* and explain what the neural network is doing layer by layer.

We can classify neural networks by the number of layers, however there are no absolute boundaries for this classification. Broadly speaking there are *deep neural networks* with 3 or more hidden layers [34] and there are *shallow neural networks* with less hidden layers. With an increasing number of layers and more nodes per layer, a neural network is able to learn more complex problems [14, 13]. A so-called "*perceptron*" is the simplest form of an artificial neural network consisting of only a single neuron [33].

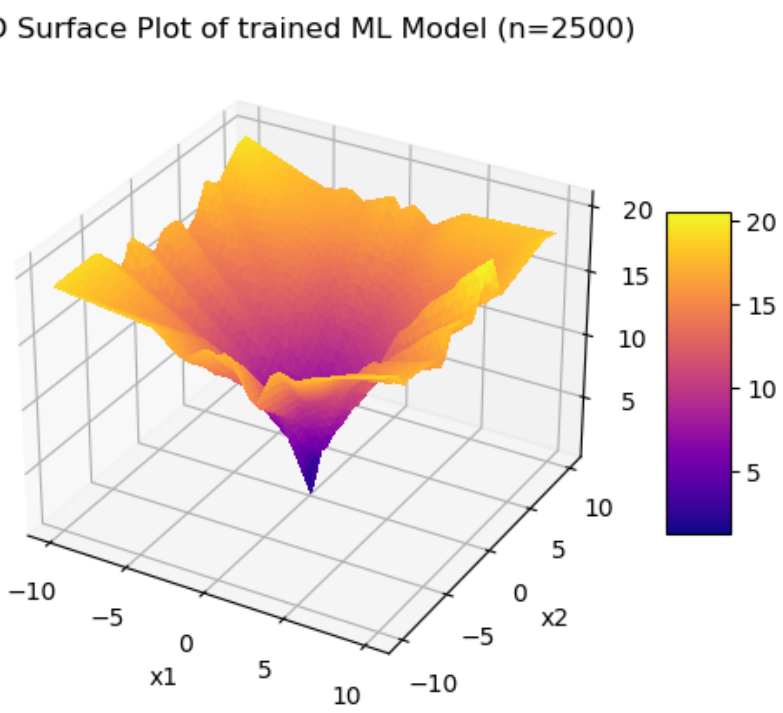


Figure 3.2.: Surface plot of the Ackley test function (Section 6.1.1) approximated by a neural network with a single hidden layer of 32 neurons. This figure visualizes the landscape of a trained machine learning model. It was fitted on a total of 106 datapoints. The model was evaluated on 2,500 points in order to plot it with fine resolution.

3.2. Models

The learning progress is stored as a *model*, which represents the algorithm's knowledge. Some models are very comprehensible, e.g., a linear regression model, which fits a straight line through the data. Such models have good interpretability and usually machine learning libraries let us access the function components to analyse it in more detail. Other models are much harder to interpret and sometimes work more like black-box functions without careful investigation, e.g., neural networks.

To visualize the information in a fitted model we can evaluate it in various positions and then plot the results. Then we can make a landscape plot of its knowledge as seen in Figure 3.2.

3.3. Motivation

There are several advantages to this modelling approach. First, machine learning algorithms excel at modelling complex relationships between input and output space that may not be recognizable for humans. The evaluation of the trained algorithm is computationally cheap, which allows us to make fast predictions of the underlying function value at some input position x. Further, if the trained model has been saved to a file, we can not only reuse it for prediction, but also train the same model again on new labeled data to improve its predictive accuracy.

Finally, the learning process operates with remarkable efficiency when implemented within an automated framework, minimizing the need for extensive manual intervention compared to other modeling methodologies. The interested reader may want to look up some of the machine learning communities active on internet platforms such as Kaggle [1] and Knime [2]. Also there is a large number of software packages for the development of machine learning applications available such as scikit-learn [3] [31], PyTorch [4] and TensorFlow [5] [32] available online.

3.4. Limitations

Machine learning algorithms highly depend on the quality and amount of initial training data. Introducing bias into the training data can have a detrimental impact on the final machine learning model. Therefore, it is crucial to ensure diverse and representative data sampling in order to mitigate bias in the training data. Some of these mitigation measures include

- selecting comprehensive and diverse data sources,
- bias assessment of the initial data to identify any under-represented or over-represented groups or characteristics,
- proper random sampling to reduce the likelihood of introducing bias,
- data augmentation to artificially create more diversity,

Moreover, while the computational cost of model predictions is negligible, the training of machine learning models is expensive. However, modern hardware drastically reduces training time when working on computers that use GPU's or TPU's to boost calculations.

Another issue is the interpretability of the final model. Machine learning algorithms often generate models without explicit, human-readable instructions. Therefore, the resulting model is often regarded as a "black box", requiring thorough examination for a comprehensive understanding of its inner workings.

[1] https://www.kaggle.com/
[2] https://www.knime.com/
[3] https://scikit-learn.org/stable/
[4] https://pytorch.org/
[5] https://www.tensorflow.org/

4. Meta Modelling

This chapter introduces the concept of meta models and how they can be used. They are also referred to as "surrogate models", because they can be used *in place of* a complex mathematical function to approximate it. First, a section about regression modelling sets the stage for the model type used in the active learning scheme studied in this work. Then, the most important performance metrics for regression analysis are explained. Second, meta modelling is presented along side some use cases, limitations and requirements. As the surrogate modelling approach is driven by data and relies heavily on the initial information, different data sampling methods are described later in this chapter. These methods are crucial, because they sample the model's ground knowledge.

There exists a common principle in modelling and analysis, often referred to as the "principle of parsimony" or "Occam's Razor." This principle suggests that when constructing a model or theory, one should strive for simplicity while still maintaining the necessary level of accuracy. We can put this idea in simpler words: "*A good model is as complex as necessary and as accurate as possible*".

4.1. Regression Modelling

In this thesis the active learning scheme is built around a regression-based meta model that can predict values of continuous variables. Therefore, this chapter starts by putting forward some general topics about regression modelling.

Regression analysis is a statistical modelling technique applied for predictive modelling. It studies the relationship between a dependent variable (also known as "target") and independent variable(s). The technique of regression analysis is essential for data analysis and many modelling tasks. There are several types of regression analysis to choose from depending on the problem at hand. These include linear regression, logistic regression, polynomial regression, ridge regression, lasso regression, etc. [35]. Some of these exist in machine learning frameworks as well, i.e., scikit learn [31] offers many modules that implement learning algorithms for this purpose that can be found on their online documentation for supervised learning [1].

Usually predictions from regression models are not accurate and they do not need to be. However, it is valuable to know the prediction error or in other words the *performance* of the model for several reasons, e.g., to estimate the prediction's confidence. We can

[1] https://scikit-learn.org/stable/supervised_learning.html

determine a predictive model's performance using one of many *performance metrics* which are equations designed to measure how close a predicted outcome is to the real quantity. The same holds for machine learning models that arise from regression-based learning algorithms, for which sci-kit learn also provides a lot of functionality [2].

4.1.1. Data Splitting in Machine Learning

The prediction error is computed using a *test dataset* which can be a fraction of the original dataset, or a completely new set of examples independent from the initial training dataset.

Typically, the dataset used to train a machine learning model is split into three distinct subsets. Firstly, the **training set**, encompassing the majority of the data (about 70-80%), acts as the learning foundation for the model. By exposing the model to this substantial portion, it refines its parameters to make accurate predictions.

The **validation set**, comprising a smaller but significant portion (about 10-15%), comes into play during the training phase. This set is useful in fine-tuning the model's hyperparameters and in preventing overfitting, ensuring that the model generalizes well to new data.

Thirdly, the **test set** (about 10-15%), kept entirely separate during training and validation, serves as an independent benchmark. Its purpose is to assess the model's real-world performance on unseen data, providing a reliable measure of its generalization capabilities.

When a model is trained properly it has a low prediction error but does not **overfit**. Otherwise, the model is not able to generalize on new data, i.e. the test dataset. Thus, measuring the prediction error may help to avoid overfitting, because an error close to zero hints at an overfitted model. At the same time we want to minimize the prediction error to the point, where we can obtain confident predictions from the model [14].

4.1.2. Performance metrics

Some of the most commonly used performance metrics for regression analysis are presented in the following. It is important to note that the formulas given below are intended for strictly positive valued observations and are not designed to handle any zeros in the observations or the predictions [35].

The following **notation** will be used (adopted from Plevris et al. [35]):
Let N be the number of samples in the dataset, p an $N \times 1$ vector representing the values of the predicted outcome, and let r be an $N \times 1$ vector that holds the real observed values of a quantity that was computed or measured.

[2] https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

Mean Absolute Error (MAE) measures the mean of the absolute error between the predicted and real values. In other words, it represents the average of the residuals $(p_i - r_i)$ in the dataset. The MAE is robust to outliers, meaning it does not easily get distorted by extreme values and is defined as

$$MSE = 1/N \sum_{i=1}^{N} |p_i - r_i| \tag{4.1}$$

Mean Squared Error (MSE) is the average squared difference between the real and predicted values in the dataset. While the MSE is a popular metric, it is prone to outliers, meaning extreme values can skew the error score. It is given as

$$MSE = 1/N \sum_{i=1}^{N} (p_i - r_i)^2 \tag{4.2}$$

Root Mean Squared Error (RMSE) is more robust to outliers than MSE and also produces an error value of the same unit like the target variable. That is why the RMSE is widely used to compare different regression models. It is defined as

$$RMSE = \sqrt{MSE} = \sqrt{1/N \sum_{i=1}^{N} (p_i - r_i)^2} \tag{4.3}$$

Coefficient of Determination

To define the Coefficient of Determination we distinguish between the linear and the nonlinear regression case. The two error measures are considered equivalent.

For the linear case we remember the general form of the linear regression in the case of two variables which can be defined as

$$\hat{p} = a + b \cdot r_i + \epsilon \quad , \tag{4.4}$$

where $\hat{p}$ is the dependent variable, r_i is the independent variable, a is the intercept, b is the slope and ϵ is some error. In other words, given the real (target) values r_i and the predictions of our original model p_i, we find that $\hat{p}$ is the best prediction obtained from another model, which is the linear regression model [35].

The **Coefficient of Determination in linear regression** R^2 represents the ratio of the variance that can be explained by the linear regression model and the total variance. A high $R2$ value indicates a well performing model. R^2 is given as

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (\hat{p} - p_i)^2}{\sum_{i=1}^{N} (p_i - \bar{p_i})^2} \quad , \tag{4.5}$$

where $\bar{p_i}$ is the mean value of p.

The **Coefficient of Determination in nonlinear regression** $R_{squared}$ is deliberately named different to avoid confusion. $R_{squared}$ is equivalent to R^2 (Equation 4.5), but it yields values in $(-\infty, 1]$, which may misguide machine learning model development. The formula below defines the general case of the Coefficient of Determination which is applied in nonlinear regression models [35].

$$R_{squared} = 1 - \frac{\sum_{i=1}^{N}(p_i - r_i)^2}{\sum_{i=1}^{N}(r_i - \bar{r_i})^2} \quad , \tag{4.6}$$

where $\bar{r_i}$ is the mean of all real observed values r.

Summary of Performance Metrics

To conclude this topic, the performance metric should be chosen depending on the use case, the expected probability of outliers and also considering what property of the model should be conveyed. Generally, the smaller the error metrics the better the model performs, in contrast to the Coefficient of Determination. The MAE can be interpreted easily, but is less versatile to use in mathematical operations than MSE and RMSE, as the latter are differentiable. MSE penalizes large errors more harshly than the other metrics. RMSE is widely used as guiding metric by machine learning engineers [35].

4.1.3. Limitations

It is worth pointing out, that these descriptive model performance metrics alone may also fail terribly, as the "*Anscombe's quartet*" shows. Anscombe's quartet is a set of four datasets that have identical simple descriptive statistics, including mean, variance, correlation, and regression coefficients, yet exhibit distinct patterns when graphically represented. Figure 4.1 illustrates how different these four data sets look when plotted. Created by the statistician Francis Anscombe in 1973, the quartet serves as a powerful illustration of the limitations when relying only on summary statistics and emphasizes the importance of visual exploration in data analysis [36].

4.2. Meta Models

4.2.1. Motivation

Engineers nowadays across many different disciplines, e.g., the automotive or semiconductor industries, but also in the design of permanent magnets, face the challenge of optimizing computationally expensive mathematical models. Their goal is finding the best solution and fine-tuning existing ones. For instance the simulation of a car crash can take about 20 hours. On the one hand, these computer models enable us to very accurately explore new design possibilities without having to manufacture expensive prototypes and conduct real-life experiments. On the other hand, the optimization of these models is a very time consuming task, because optimization algorithms require

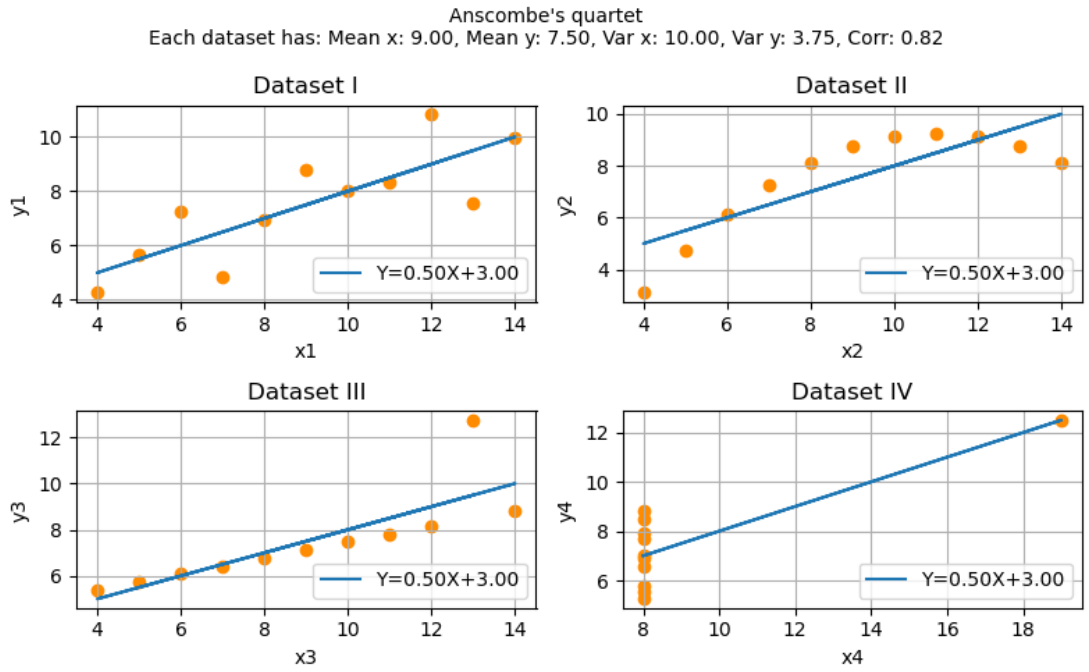


Figure 4.1.: Anscombe's quartet: For all four data sets the mean (Mean), standard deviation (Var), and the Pearson correlation coefficient (Corr) are identical. Nevertheless, the graphical representations of the data sets are different.

many function evaluations. Additionally, the task becomes even more challenging for complex, high-dimensional problems [2].

4.2.2. Usage

The optimization process of such large models can be accelerated greatly by decreasing the number of function evaluations. To obtain a rough approximation of the objective space, we can model a *response surface* by fitting a model with some existing datapoints which act as supporting points that inform it about the function domain. There are a few possibilities to build such a model, e.g., with splines, polynomials or using machine learning. This approach is also called *meta modelling* [37, 16, 2]. The choice of model is important in order to describe as much complexity as possible.

Jones and Schonlau [2] describe a method known as *Efficient Global Optimization (EGO)* algorithm (Section A.1). Essentially, this algorithm exploits the cheap function evaluations of the meta model to guide the optimization process. This meta modelling approach is often used in engineering problems.

4.2.3. Limitations

As a data driven method the meta modelling approach heavily relies on the initial data used to calibrate the model. There is a trade-off in the size of the initial dataset: On the one side, if the sampling size is too small, the meta model becomes very inaccurate and may miss important regions. Then, the search for the optimum becomes inefficient. On the other side, sampling many data points is very expensive [2]. The sampling quality is also vital for the efficiency of this approach. The initial sample should cover as much space as possible and as evenly as possible. The quality of the sample can be measured in the form of a *discrepancy* metric (Definition 1) suggested by Ilya M. Sobol'.

Definition 1 (Discrepancy [38]) *Let there be a hypercube with N points inside. (A hypercube is a multi-dimensional generalization of a cube. It is a geometric figure with intriguing properties for theoretical study and practical applications.) The theoretical density of this hypercube is* $d_t = 1/N$. *Now, imagine an arbitrary hyper-parallelepiped* P_i *that is constructed within this hypercube. (A hyper-parallelepiped is similar to the hypercube a generalization of a cube with an arbitrary number of dimensions. However, the cube that it resembles is bounded by parallelograms instead of squares.) The N points in the hypercube are not partially also inside the hyper-parallelepiped* P_i. *Thus,* P_i *now has its own point density* d_i. *Then, the discrepancy is defined as the maximum deviation between* d_t *and* d_i.

Of course we can plot datasets to visually evaluate the quality of the sample. However, in high dimensional search space ($k_{\text{in}} > 3$) [16] this is not possibly anymore. So, we rely on the discrepancy metric. For low dimensional decision space ($k_{\text{in}} <= 3$) simple methods like grid sampling or random sampling are good enough. However, in high dimensional decision space more sophisticated sampling methods are required to obtain a low discrepancy sample with statistical guarantee while keeping the sample size reasonably small.

4.3. Sampling Methods

This subsection outlines four sampling methods to draw a predefined number of "random" samples from a designated search space. Each one has special advantages and limitations. The plot of Figure 4.4 illustrates the nature of each sampling method.

4.3.1. Grid Sampling

Grid sampling is a structural method, where first a point grid that covers the entirety of the spatial domain is established. Then, points from this grid are selected in predefined intervals. Due to its structured nature this approach will comprehensively explore the entire search space. In specific applications, such as finite element methods, employing grids often simplifies algorithmic procedures when compared to more irregular point sets. However, in high-dimensional search spaces, this technique will generate large, systematic

pockets (where the domain is left unexplored). If these pockets need to be reduced or if the search space is high-dimensional ($dim > 3$), the sample size N explodes and sampling becomes infeasible. For example if we want to sample $n = 10$ points per dimension in a search space of dimension $dim = 5$ the sample size given by $n^{dim} = N$ would be $10^5 = 100,000$. However in a 2-dimensional search the same interval $n = 10$ would result in a sample size 100. When looking at Figure 4.2 we can see the systematically occurring pockets between the points sampled in blue.

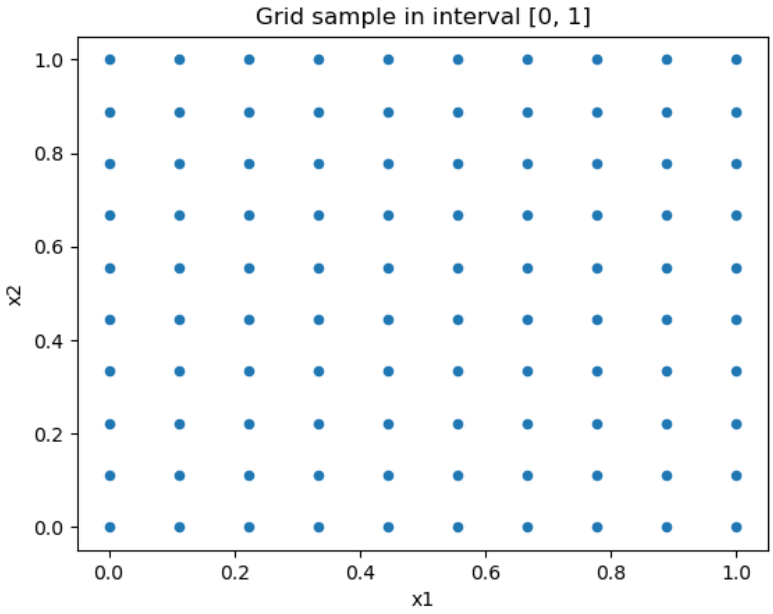


Figure 4.2.: Plot of points sampled on 2-dimensional grid with 10 points in each dimension.

4.3.2. Pseudo-Random Sampling

Pseudo-Random sampling is a straightforward method to create a sample of a given size N. In computational experiments usually a pseudo-random number generator (available in many software packages), draws N points, i.e., from a uniform distribution U in the interval [0,1]. When drawing from the uniform distribution each point is equally likely to be drawn. So, in this example, each random point X_i is given by $X_i \sim U(0,1)$ *with* $i = 1, 2, ..., N$. This method is cheap and simple, but cannot guarantee a low discrepancy sample from a high-dimensional search space due to its random nature. Typically some points are clustered and/or large pockets occur in the search space [39].

4.3.3. Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) works similar to *stratified sampling* (see definition by Burhenne et al. p. 1817 [39]). Essentially, stratified sampling works by defining a number of intervals for all dimensions. Then, each interval is divided into subintervals, of which each holds a certain number of samples. Now, the unique feature of LHS is that, each sampled point is associated with exactly one interval of each dimension. This special property is illustrated in Figure 4.3. The plot in this figure shows two highlighted sample points in red and blue to illustrate how this sampling method works.

LHS works best in high-dimensional search spaces and is very efficient sampling with low discrepancy [39]. In a Python environment we can import an implementation of the sample from the SciPy library [3] [40]. Also, there is the option to use a post-processing step after sampling that runs an optimization scheme which improves the quality of the sample.

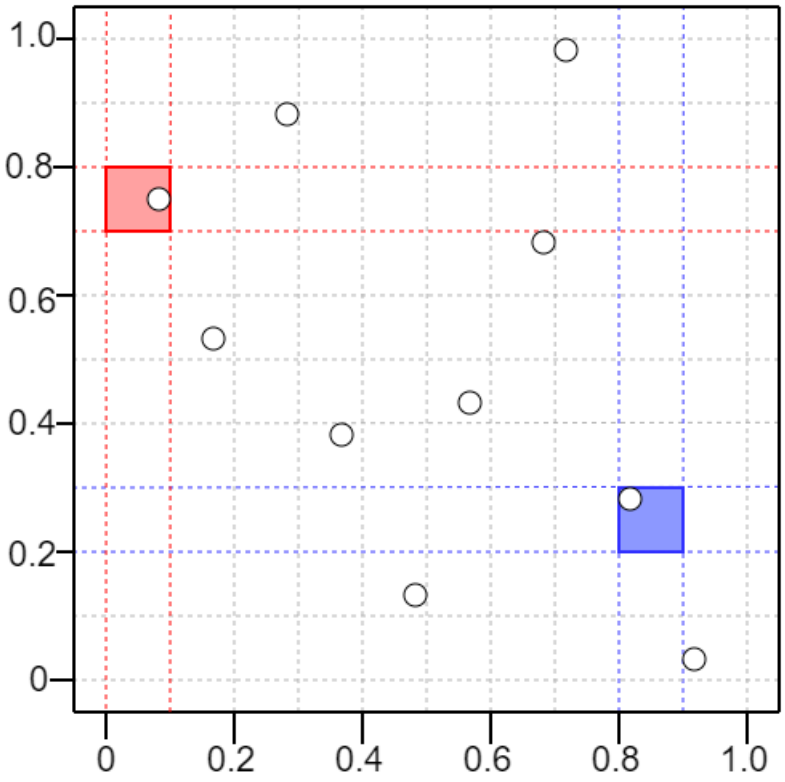


Figure 4.3.: Plot of a Latin Hypercube sample. The black circles are points sampled following the rules of LHS. The dashed lines mark the boarders of the intervals. The red and the blue zone highlight how the two sample points are unique in their respective intervals.

[3] https://docs.scipy.org/doc/scipy/index.html

4.3.4. Sobol' Sequences

In 1967, Ilya M. Sobol' introduced an efficient algorithm to generate *Sobol' sequences* with low discrepancy [41]. These sequences are part of the quasi-random sequences which are designed to sample in a multi-dimensional search space as uniformly as possible. The Sobol' sequence is mathematically constructed as explained in the paper by Sobol' et al. p. 65 [41].

Quasi-random numbers differ from pseudo-random numbers mainly due to the fact that each point is chosen after considering the previously sampled numbers. Thus, sampling with quasi-random numbers prevents gaps and clusters [39].

In a Python environment we can import the package [4] to generate Sobol' sequences. To get a statistical guarantee of the sequence's balance properties, we can draw $N = 2^m$ (base 2) points from this sequence generator. The SciPy library also offers to use a post-processing step after sampling that runs an optimization scheme which improves the quality of the sample [40].

4.4. Summary of Sampling Methods

To sum up the information about the four different sampling methods, we will take a look at the 4x4 plot of Figure 4.4. This plot visualizes and compares the nature of each sampling method (columns) across the sample sizes (rows) 5, 10, 50, 100 in two dimensions (x1, x2). Below each subplot we see the sample's discrepancy.

We can observe a trend from left to right: The two sampling methods on the right (LHS and Sobol' sequence) have a lower discrepancy than the structural grid sampling and the pseudo-random sampling approach. We can confirm visually that the samples of columns three and four are less clustered and have smaller gaps than the first two columns. This effect is better visible in larger sample sizes. While we are able to analyse these two-dimension plots visually, we are dependent on the discrepancy metric for higher dimensional samples.

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.qmc.Sobol.html

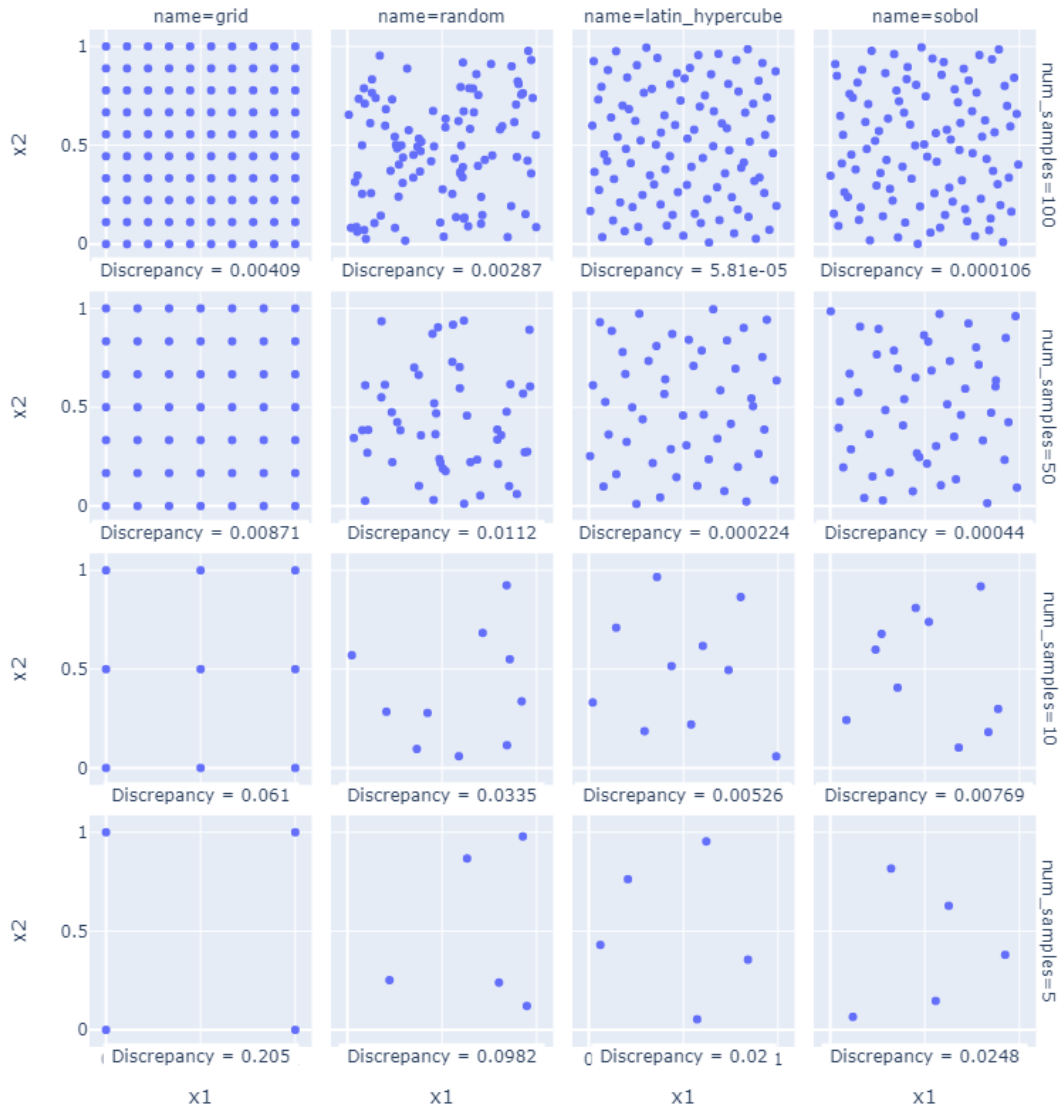


Figure 4.4.: Comparison of sampling methods
This 4x4 plot visualizes the comparison of different sampling methods. The columns from left to right show an example of grid sampling, pseudo-random sampling, Latin hypercube sampling and Sobol' sequence sampling. The rows represent the sample size from bottom to top 5, 10, 50, 100. Below each subplot we see the sample's discrepancy. We can observe a trend from left to right: The two sampling methods on the right (LHS and Sobol' sequence) have a lower discrepancy than the structural grid sampling and the pseudo-random sampling approach. We can confirm visually that the samples of columns three and four are less clustered and have smaller gaps than the first two columns. This effect is better visible in larger sample sizes.
This figure was plotted with the source code of Baird and Sparks [42].

5. Active Learning Scheme

This chapter explains what an active learning scheme (ALS) is, by joining the theory of the preceding chapters, as they set the stage for the fundamental concepts of this scheme.

First, this chapter will discuss how to characterize the active learning scheme and how the Efficient Global Optimization (EGO) algorithm is associated with this scheme. Their differences and similarities will be described, as well as their areas of application. Second, the general architecture of an ALS is illustrated and outlined. Afterwards, possible outcomes of the active learning scheme are presented. Finally, the motivation for usage of active learning schemes and possible limitations are explained.

5.1. Characterization

The "active" learning scheme is a general learning framework as opposed to purely "passive" machine learning (Section 3.1.2). The ALS can be considered "active" because the scheme is able make choices, while the machine learning algorithm is essentially passively experiencing the data it is provided with. According to Settles [12], the core hypothesis of the ALS is that a machine learning algorithm (Section 3.1), that can select the data from which it learns autonomously, achieves better performance with less training compared to a passively learning algorithm. The active learning scheme can be categorized as a subfield of machine learning that may be put to use in various problem types, e.g., regression tasks or classification tasks. The key idea in active learning is that the learning algorithm is able to "ask questions", and thus it guides its own learning process. One might argue that the algorithm is allowed to be "curious" [12]. Figure 5.1 illustrates this learning behaviour.

The EGO algorithm (Section A.1), suggested by Jones et al. in 1998 [37], generally follows a similar idea as the active learning scheme and, as the name suggests, it is utilized solely for optimization. While the EGO algorithm typically does not use machine learning models and relies on an expected improvement function, it can be seen as a related framework. Thus, the core idea of the ALS has existed for quite some time. To point out some similarities, both EGO algorithm and ALS train a predictive model iteratively in order to increase their performance while keeping the number of expensive validations low. The EGO algorithm only aims at finding the optimum, while the active learning scheme can have other objectives as well.

5.2. General Architecture

Figure 5.1 visualizes the general architecture of an active learning scheme. At the center there is a circular pipeline that powers the learning process of the *Learner*. Each iteration the *Learner* asks the *Oracle* for *labels* it wants to know to increase its knowledge and thus its predictive ability. The number of labels the *Learner* wants to know is defined by the *learning rate* lr. In other words, lr describes the number of validated results that are added to the knowledge of the *Learner* in each iteration. The *learning rate* can be interpreted as a hyperparameter in machine learning, which can be tuned to suite the task. If lr is high, the amount of computationally demanding validations is higher as well and the *Learner's* knowledge grows faster in each iteration. The goal is to pick the lr as low as possible, to minimize the number of validations. The learning process may start when the *initial labeled dataset* is provided. Finally, the loop is broken if a stopping condition is met. At the end of the scheme a trained model with according training data persists. This labeled training data is partly from the initial sample and partly defined by the *Learner's* queries. Depending on the *objective* and the stopping criterion other useful properties are yielded as well.

5.3. Motivation

In general, the ALS is best applied in supervised learning problems. The ALS is efficient, when the acquisition of labels is difficult or expensive and unlabeled samples are abundant. Each iteration of the scheme determines a small set of samples to label, which promises to boost the model performance [12].

As explained in Section 4.2.1, high-fidelity physics simulations are difficult to optimize, but indispensable in many fields of engineering. Employing meta models during the optimization process can drastically decrease the computational cost, as the EGO algorithm (Section A.1) shows.

Recently, machine learning algorithms (Section 3.1) were successfully applied for surrogate modeling (Chapter 4) and play an essential role in active learning schemes that were used in research [10, 11, 12, 3]. Furthermore, the development in computer hardware enables and improves the efficient use of machine learning algorithms. There is a whole range of versatile and flexible machine learning algorithms available to model various complex, multimodal and non-linear functions for simulations.

Possible Outcomes

What results does the active learning scheme offer? The ALS can be used for optimization, if the Learner's objective is that of the objective function. The Learner's check would then define queries that explore domain areas that are closer to the expected optimum. The scheme can also be utilized to establish a model that is overall well informed without any large "knowledge gaps". This is possible if the Learner chooses to query for samples in hardly explored areas to find a good coverage of the data. Several other areas of application are described in Settles' 2009 report on the active learning scheme [12].

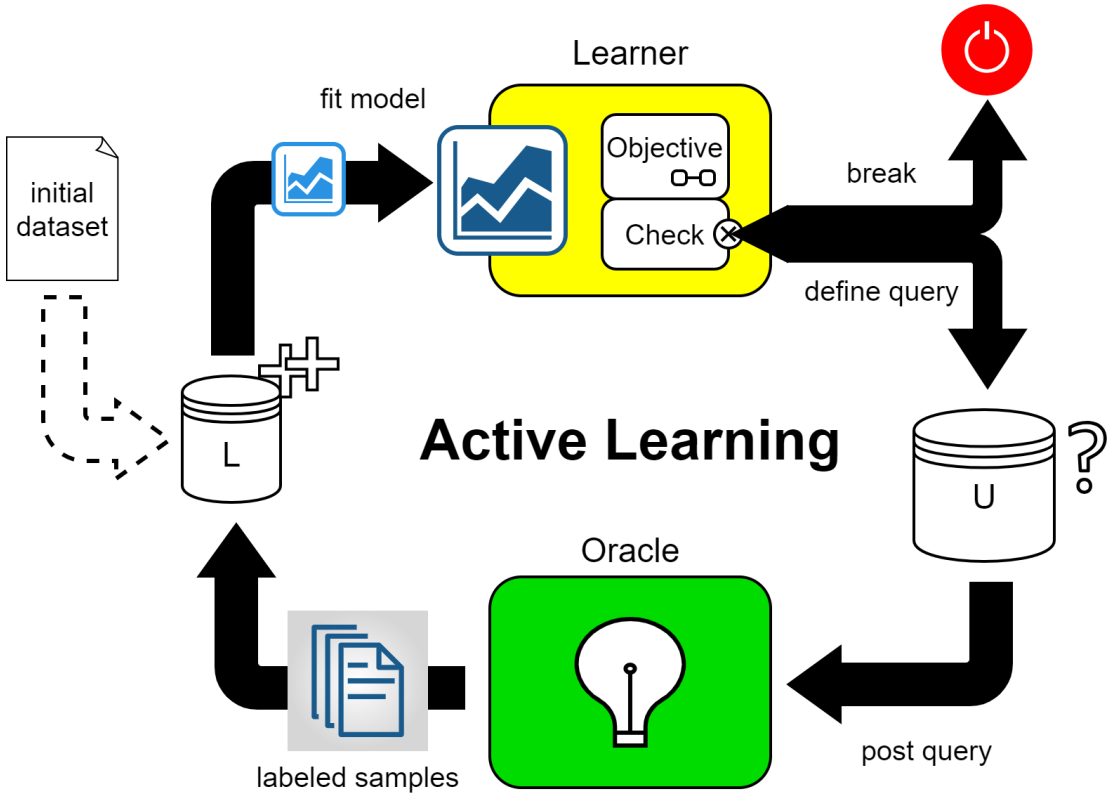


Figure 5.1.: General flow of active learning scheme
This flowchart illustrates how an active learning scheme is generally set up. To start the cycle we integrate an **initial dataset** as the basis for the labeled data storage which contains real measurements or observations. Here, this data storage is denoted "L" for **labeled data** and grows in each iteration. A **model** is fitted to the labeled data and passed on. The **Learner** knows the objective of the scheme. It receives the fitted model and performs a check to see if the model meets the objective or if some termination criterion is met to break the cycle. If the Learner decides to continue the cycle, it defines a **query** that represents the areas of the domain that should be explored next. This query consists of a number of **unlabeled samples**, marked with "U" and is posted to the Oracle. This number is defined by the **learning rate** lr. The **Oracle** is able to annotate the labels on the lr unlabeled samples from the query. Then, the freshly labeled samples are fed back into the labeled data storage increasing the total amount of labeled data by lr datapoints. Thus, the cycle is complete and the next model is trained on more data and is therefore better informed and probably performing better than the previous one.
This figure is an adaptation from Settles [12] visualization of the ALS pipeline.

Example use-case in micromagnetic simulation

One excellent example of an active learning scheme application is the work of Kovacs and colleagues titled "*Physics-informed machine learning combining experiment and simulation for the design of neodymium-iron-boron permanent magnets with reduced critical-elements content*" [3]. Essentially, they have set up an automated active learning scheme with a *Learner* that consists of a shallow neural network as surrogate model and a genetic algorithm to locate promising candidate solutions. The *Oracle* that validates and labels the proposed solutions is a micromagnetic simulation that takes several hours to annotate a single candidate. Using this setup they explored new possible chemical compositions for permanent magnets with the goal of maximizing the magnetic property *coercivity* [1], and minimizing the *neodymium content* [2].

5.4. Limitations

As the training data is not drawn independently and identically from the underlying data distribution by the learning algorithm, we cannot simply apply another machine learning algorithm to the same data, which is useful in certain tasks. So, data sampled by one scheme cannot be reused in another scheme setup.

Further, in some cases the active learning scheme requires more labeled data than the same supervised learning algorithm in a passive implementation. There is no guarantee, that the ALS is more efficient than passive learning algorithms [12].

[1] Coercivity quantifies the resistance to demagnetization of a ferromagnetic material [8].

[2] Neodymium (Nd) belongs to a group of rare chemical elements and it is crucial for the performance of permanent magnets. Reducing the amount of Nd in sufficiently powerful magnets helps lowering the industry's required yearly amount and opens possibilities for usage of more environmentally friendly alternative chemical compositions [3].

6. Parameter Study

This chapter explains how the performance of he ALS is studied. The first half will outline the experimental setup and explain how the comparative parameter study is carried out. In the second half, we take a look at the modular python framework which was developed to setup a system or engine that performs the optimization with a common tool chain for experiments with the active learning scheme and also for optimization employing only an optimization algorithm.

6.1. Experimental setup

Essentially, the two optimization approaches A) solely optimization algorithm and B) active learning scheme are compared with the same configuration on certain test problems with known solutions.

This study focuses on the following aspects:

1. The amount of overall computational effort to find an optimal solution
2. The influence of different sampling methods on both optimization approaches
3. The influence of initial sample sizes on both optimization approaches
4. The meta model performance score

Ad 1.: The number of "original" objective function $f_{orig}(\cdot)$ evaluations is compared in both approaches. The number of "model" evaluations $f_{model}(\cdot)$, meaning computation of points in the meta model are computationally cheap enough to dismiss their contribution to the overall computational cost. In other words the number of $f_{model}(\cdot)$ is negligible, however the validation phase of the active learning scheme requires some calls to $f_{orig}(\cdot)$, which are counted.

Ad 2.: The sampling methods are utilized for the initial population sampling of the genetic algorithm used for optimization. Additionally, the sampling methods mimic real-world measurements that are required as training data for the model in the active learning scheme.

Ad 3.: The sample size determines the size of the initial population in the genetic algorithm and the size of the training data set for the surrogate model.

Ad 4.: A precondition to use the surrogate model is a high model score of $R^2 > 90\%$. Then we can trust its predictions and that the model will guide the active learning scheme with enough confidence.

The experiment was conducted on the software described in the following Section 6.2. The program was configured with the configuration parameters listed and described in the Appendix Section A.2. For this parameter study the following configuration parameters acted as control variables:

- The optimization approach
- The test function
- The sampling method
- The sample size

The parameter choices are described in more detail in the following subsections.

To measure the convergence of each optimization approach in single-objective problems the Euclidean distance between the computed optimum and the true optimum in objective space $d(f(\hat{x}), f(x^*))$ (Equation 2.2) was used. For multi-objective problems the IGD (Equation 2.3) was utilized.

The common convergence goal for both approaches was an error or distance of 0.05 in objective space, measured in $d(f(\hat{x}), f(x^*))$ or IGD respectively.

6.1.1. Choice of test problems

The experimental setup requires known test problems from the field of optimization problems. Obviously, it is easy to determine the performance of two approaches, when they are applied to the same problem with a known solution to contrast their outcomes. Thus, a few test functions from the domain of single and multi-objective optimization of different complexity with known Pareto sets are used. For all test problems the pymoo [25] implementation was used.

Simple Quadratic

The *Simple Quadratic* test problem is a smooth unconstrained single-objective function with a *convex* search space (for a more in depth explanation of *convexity* see [43]). Its minimum is located at $X^* = (x_1, x_2, ..., x_n) = (2, 2, ..., 2)$ where the objective function value $f(x^*) = 0$. Figure 6.1 illustrates the function with $k_{in} = 2$. The Simple Quadratic function will be used with 2 and 10 input variables in the experiment. The function is defined by

Objective function:

$$\min_{x \in \mathbb{R}^{k_{\text{in}}}} f(x) = \sum_{i=1}^{n} (x_i - 2)^2 \tag{6.1}$$

Here, $x = (x_1, x_2, ..., x_n)$ represents the decision variables, and n is the number of variables specified when initializing the problem. So, n is synonymous for k_{in}. The objective is to minimize the function $f(x)$.

k_{in}	k_{out}	Decision space X	Objective space Y	Characteristic
2	1	-5 < X < 5	0 < Y < 100	convex
10	1	-5 < X < 5	0 < Y < 100	convex

Table 6.1.: Key figures of simple quadratic test function

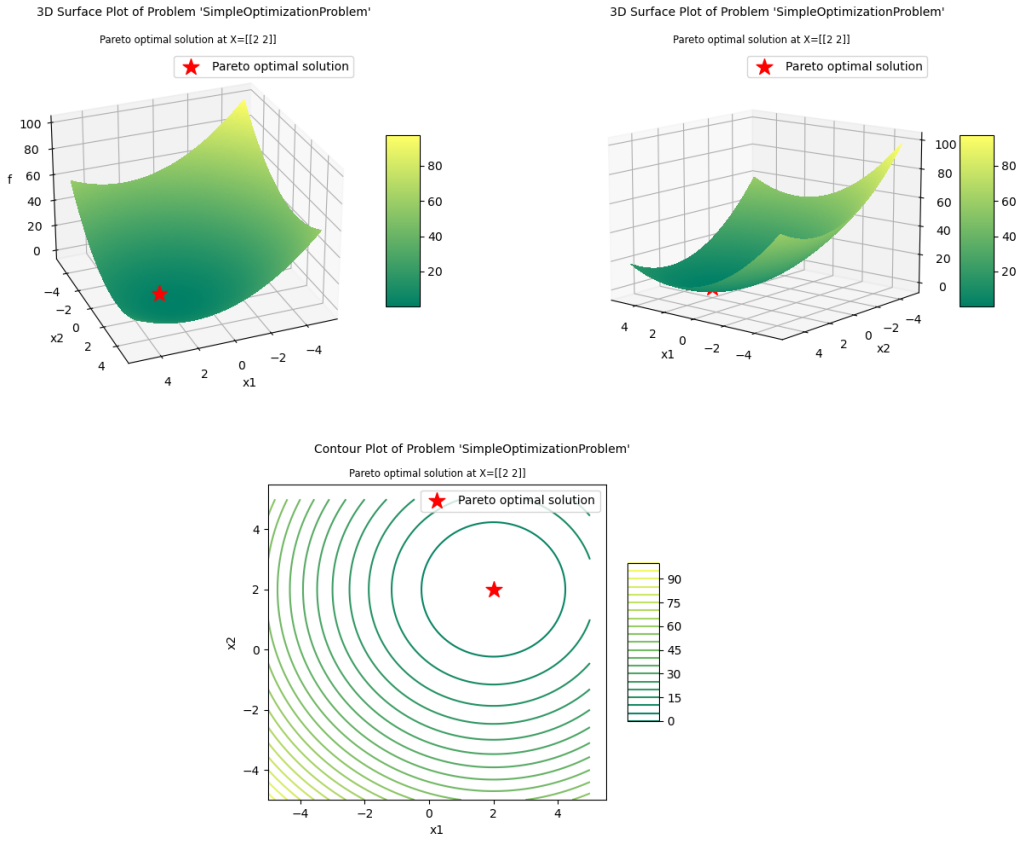


Figure 6.1.: In the upper row two surface plots of the simple quadratic test function are shown. The function value f is on the z axis while the two input variables $x1$ and $x2$ are on the x- and y-axis. The plot below shows the contour lines of simple quadratic test function, where the two input variables are on the two axes labeled $x1$ and $x2$ and the color indicates the function value. The red star marks the location of the minimum.

Griewank

The *Griewank* [1] test function is a simple but noisy unconstrained single-objective function that is *non-convex* (for a more in depth explanation of *convexity* see [43]) and thus it is hard to optimize with an algorithm that is not specialized. Its minimum is located at $x^* = (0, 0)$ where $f(x^*) = 0$. Figure 6.2 illustrates the function which looks in 3D space like a bowl with a noisy (= not smooth) surface.

k_{in}	k_{out}	Decision space X	Objective space Y	Characteristic
2	1	-600 $<$ X $<$ 600	0 $<$ Y $<$ 175	non-convex

Table 6.2.: Key figures of test function Griewank

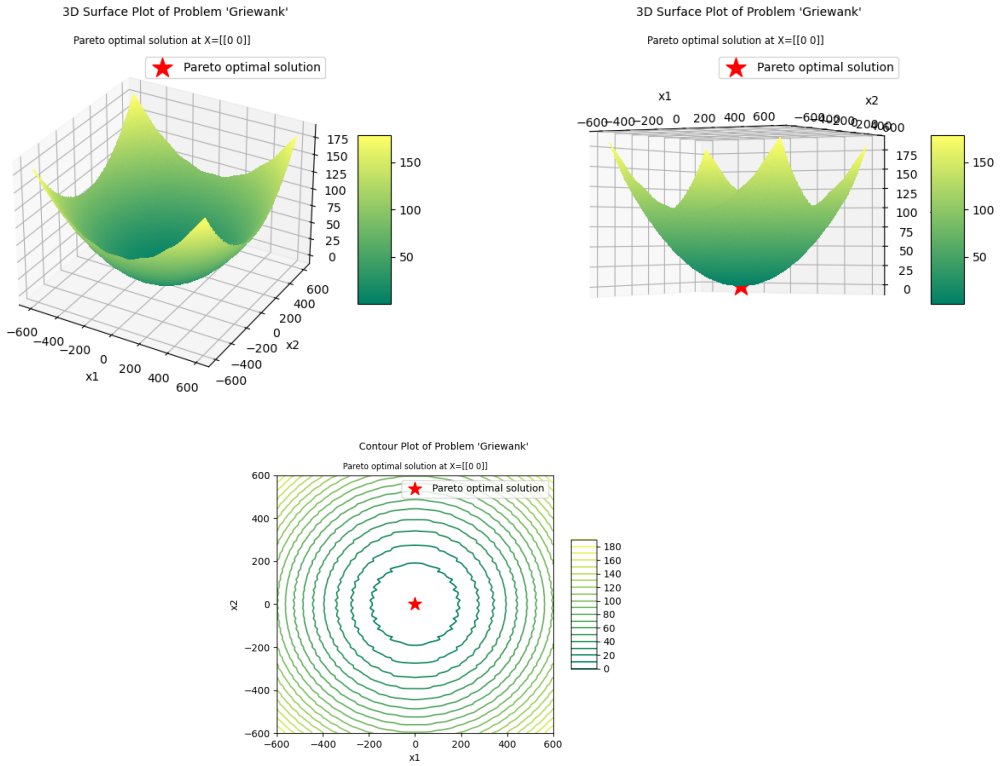


Figure 6.2.: In the upper row two surface plots of Griewank are shown. The function value f is on the z axis while the two input variables $x1$ and $x2$ are on the x- and y-axis. The plot below shows the contour lines of Griewank, where the two input variables are on the two axes labeled $x1$ and $x2$ and the color indicates the function value. The red star marks the location of the minimum.

[1] https://pymoo.org/problems/single/griewank.html

Ackley

The *Ackley* [2] test problem is a rather complex unconstrained single-objective optimization problem. It has one known minimum located at $x^* = (0, 0)$ with a function value of $f(x^*) = 0$. Figure 6.3 illustrates the function which looks like a steep funnel in 3d space. The problem is non-convex ([43]) and thus it is hard to optimize with an algorithm that is not specialized [44].

k_{in}	k_{out}	Decision space X	Objective space Y	Characteristic
2	1	-10 < X < 10	0 < Y < 20	non-convex

Table 6.3.: Key figures of test function Ackley

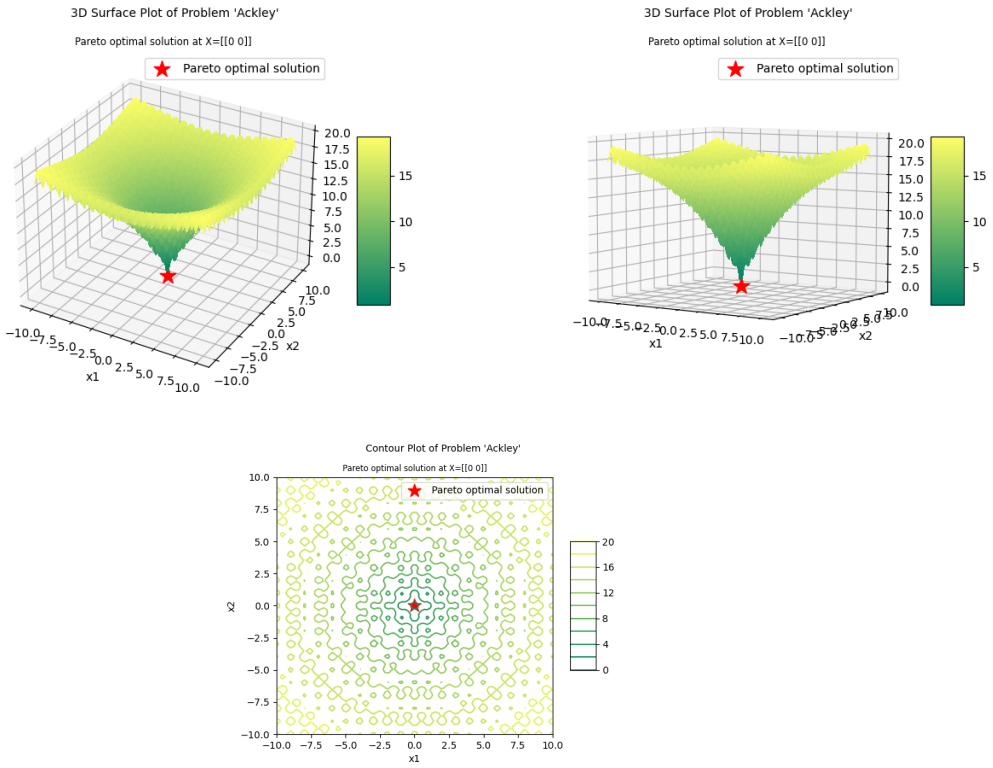


Figure 6.3.: In the upper row two surface plots of Ackley are shown. The function value f is on the z axis while the two input variables $x1$ and $x2$ are on the x- and y-axis. The plot below shows the contour lines of Ackley, where the two input variables are on the two axes labeled $x1$ and $x2$ and the color indicates the function value. The red star marks the location of the minimum.

[2] https://pymoo.org/problems/single/ackley.html

ZDT1

The *ZDT1* [3] function is a very simple multi-objective test function with a convex ([43]) Pareto set. Its decision space can be scaled up to 30 variables. In this study we will use the function on 2 and also 10 input variables [25]. Its Pareto front is illustrated in Figure 6.4.

k_{in}	k_{out}	Decision space X	Objective space Y	Characteristic
2	2	$0 < X < 1$	$0 < Y < 1$	convex
10	2	$0 < X < 1$	$0 < Y < 1$	convex

Table 6.4.: Key figures of test function ZDT1

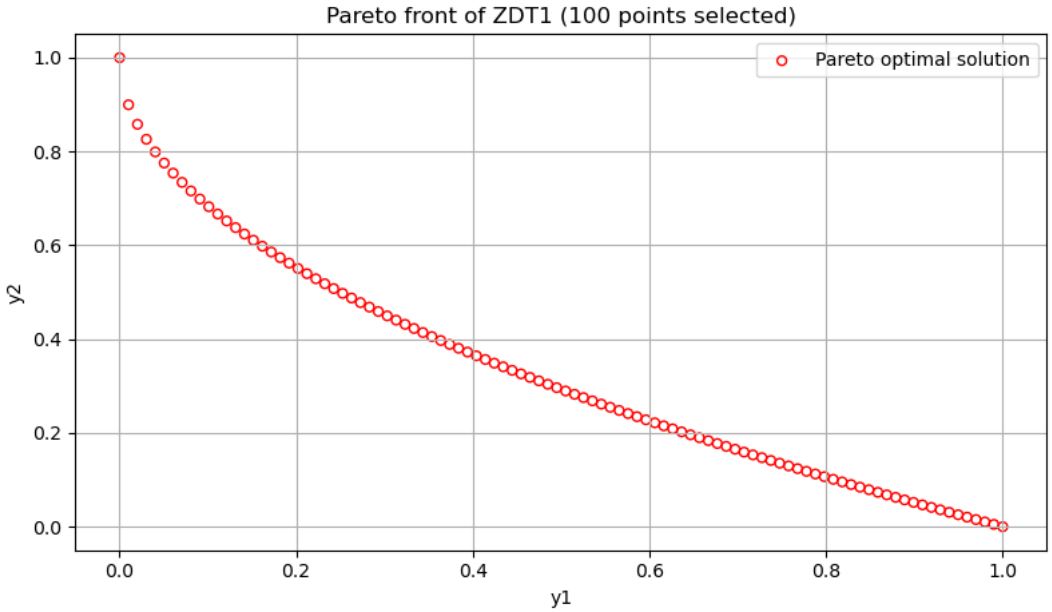


Figure 6.4.: Pareto front of ZDT1. The front looks like that for any valid number of decision variables up to 30.

6.1.2. Choice of optimization algorithm

The genetic algorithm (Section 2.4.2) NSGA-II [45] was chosen for optimization, where *NSGA* stands for "Non-dominated Sorting Genetic Algorithm". It is implemented in the pymoo library and was used in default configuration [4].

The algorithm has good convergence on a wide range of problems, but it is not very fast compared to, e.g., a gradient-based method. However, it is an overall robust algorithm.

[3] https://pymoo.org/problems/multi/zdt.html#ZDT1
[4] https://pymoo.org/algorithms/moo/nsga2.html.

6.1.3. Choice of meta model

For the meta model (Section 4.2) in the active learning scheme a *shallow neural network* (Section 3.1.4) was selected. The implementation from scikit-learn called Multi-layer Perceptron regressor (MLPRegressor) [5] was used. The MLPRegressor is a feedforward neural network which comes by default with an input and an output layer. As solver the "Limited Memory Broyden-Fletcher-Goldfarb-Shanno" (LBFGS) algorithm was used with a maximum number iterations of 4,000. The task of the solver is to optimize the weights sitting in each neuron. The LBFGS solver was selected, because it performs well on small datasets [31], which we expect to use in a real-world use case of the active learning scheme. Other hyperparameters were set on default values.

For the problems with $k_{in} = 2$ input dimensions a single hidden layer with 32 neurons was used. Even if the problem was very noisy (i.e. Griewank) the architecture was not changed, to allow for a fair comparison of the active learning scheme among problems of similar complexity. The problems with $k_{in} = 10$ variables were approximated by an architecture with 2 hidden layers of with first 256 and then 128 neurons.

The MLPRegressor was used in the ALS throughout the study and was flexible enough to learn the newly acquired data. It proved to be overall quite effective at modelling the underlying objective function.

6.1.4. Choice of initial data

Each sampling methods described in 4.3 was applied to evaluate the performance of both schemes in separate experiments. So, grid sampling, pseudo-random sampling, Latin hypercube sampling and samples from the Sobol' sequence were used. When solving solely with the optimization algorithm, the initial population of the genetic algorithm was sampled with the according method. In the active learning scheme the initial train data was generated using the respective method to simulate the initial set of measurements that would serve as training data for the model in a real world scenario. Additionally the population-based NSGA-II algorithm sampled its initial population with the method as well.

The sample size N was chosen such that they fulfill two conditions. First, N should be in base-2 to allow proper sampling from the Sobol' Sequence. Second, the sample size should be a square number to allow us to sample in 2-dimensional decision space using the grid sampling. Additionally, the sample size should be small like in a use case for the active learning scheme, where acquiring samples is expensive. Therefore, the different samples will be better comparable. Hence, in the experiments the sample size N took the values $N = [64,\ 256]$.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

6.2. Modular Python Framework

This section describes the software which was developed to help studying the active learning scheme's performance on different problems. The Framework is programmed in the language Python 3.10.

6.2.1. Purpose

In order to conduct experiments and to evaluate the performance of an active learning scheme, a common code base was developed to setup the active learning pipeline as described in Figure 5.1. To make the results comparable it is necessary to determine benchmarks and to perform the computations on one common framework. Therefore, we developed a Python framework to study the ALS in different configurations. Additionally this hands-on approach allowed us to get deeper insights into the potentials and limitations of this method.

6.2.2. Design and Features

The software development followed several goals. A major design decision was to build it modular, such that it can be arranged to solve specific problems with individual configurations. The modules are described in Figure 6.5. Another goal in development was to make the internal processes comprehensible and to allow the user to look into the internal processes which is essential for this performance study. Further, the software follows a generalized design to solve a wide range of problems. Lastly, the framework allows to switch the solving strategy for optimization problems between the active learning scheme or solely an optimization algorithm. The software consists of two main parts: *engine* and *control*.

The first part is the ***engine*** which provides the common functionality for any configuration and defines the modules that embody the entities from Figure 5.1. The modules and their interaction in a general active learning scheme are illustrated in Figure 6.5. Basically, these modules are implemented as Python classes that each play a certain role. The engine borrows a lot of functionality from the *pymoo* optimization library version 0.6 [25], because it provides a lot of tools for configuring optimization problems and it implements powerful optimization algorithms. The engine also implements an interface that allows to use the pymoo functionalities to optimize machine learning models. Functions related to machine learning are imported from sci-kit learn [31] i.e. error measures and machine learning algorithms. Additionally, some multi-objective results are analyzed using the *paretoset* package [46] to determine Pareto optimal solutions in a dataset. Lastly, the engine implements several functions to plot and analyse results implemented in the engine which are essential to determine the scheme's performance.

The second part is responsible for ***control*** and holds scripts that setup data pipelines for each scheme. Currently, there exists a pipeline for the active learning scheme and another

one for just solving with the optimization algorithm without any surrogate model. In this part of the framework the user configures the solving strategy, termination criterion, options for plotting, etc.

A few notes on the ALS software flowchart in Figure 6.5:

- In the active learning scheme during the first iteration a new machine learning model is generated. Later, this model is fitted to the the enriched data, meaning the initial dataset plus the new validated data points. The *ML Model Generator* can also tune the model's hyperparameters if required.
- The *Optimizer* solves optimization problems with the configured optimization algorithm. Its output is a vector for a promising candidate solution in decision space. The abbreviation "SOOP" means Single-Objective Optimization Problem and "MOOP" means Multi-Objective Optimization Problem
- The *Validator* applies the high-fidelity simulation to compute the true objective function value for each promising candidate. This process is supposed work in parallel in a future implementation to save work time.

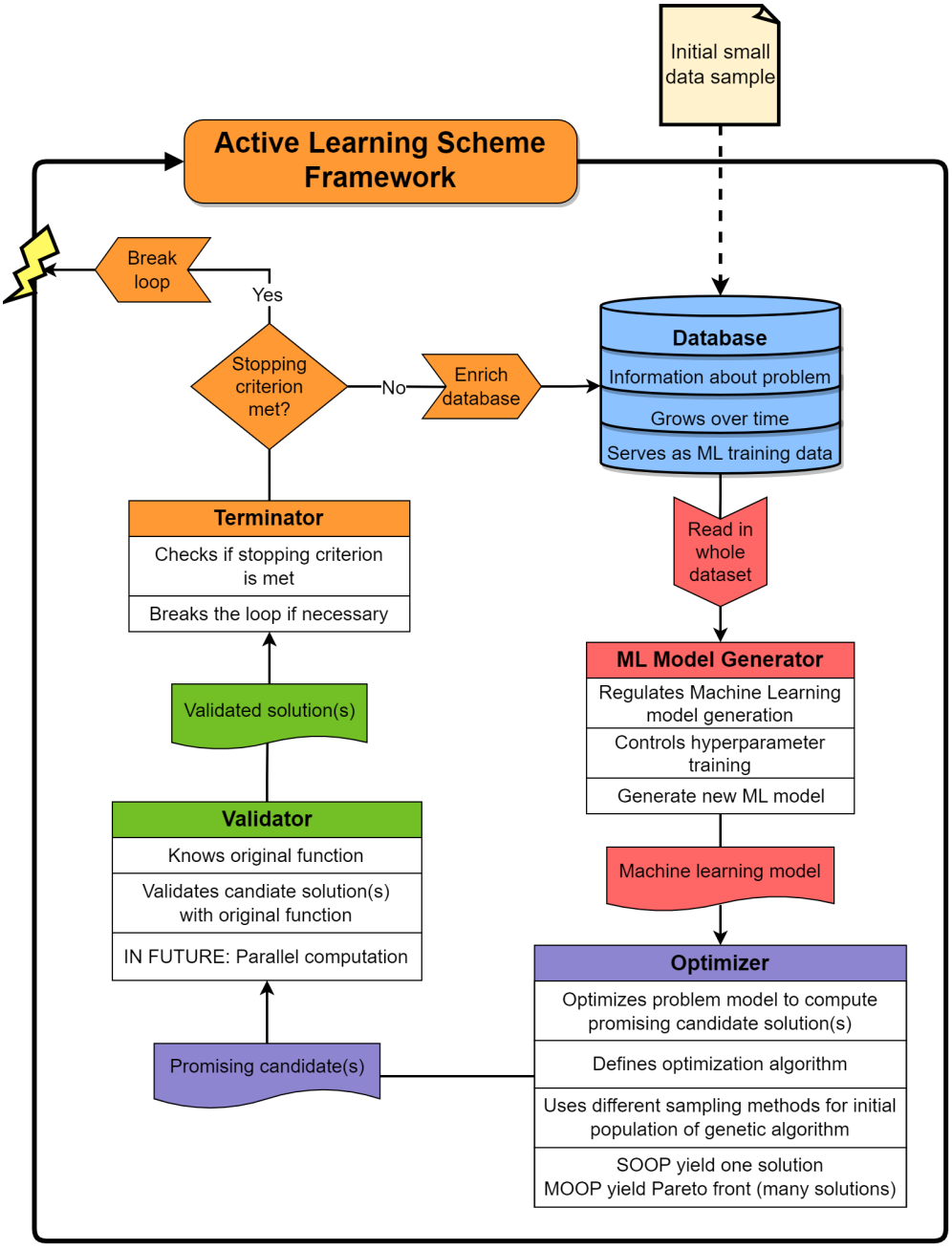


Figure 6.5.: Active learning scheme Software Flowchart
This flowchart illustrates the design of the modular Python framework on a surface level. The design is inspired by the general setup of the active learning scheme (Figure 5.1). The modules are grouped by color. So, red stands for *machine learning*, violet stands for *optimization*, green stands for *validation* and orange stands for the part that *checks for convergence*. Each module is implemented in the **engine** part of the Python framework applying object-oriented programming.

7. Results

The results of the parameter study described in the previous chapter are presented here. Each test function was solved with a combination of the variable parameters (optimization scheme, sampling size and sampling method). The experiment does not provide statistically significant results, because the differently configured experiments were not extensively repeated. The final results presented here are just a small sample of experiments and can be understood as proof-of-concept results. Thus, they highlight possible differences between the approaches and also reveal major weaknesses.

The tables display the numbers quantifying the convergence speed. The first table in each section holds the number of evaluations n_{eval} per configuration and the second table lists the number of generations of the genetic algorithm (GA) or the number iterations of the active learning scheme (ALS), respectively.

The tables are structured as follows:

- The columns represent the sampling scheme (GA for Genetic Algorithm or ALS for Active Learning Scheme) and the sample size (either $n = 64$ or $n = 256$).
- The rows denote the sampling method (GS, PRS, SS or LHS), where the sampling methods are abbreviated as Grid Sampling (GS), Pseudo-Random Sampling (PRS), Sobol' Sequence (SS) and Latin Hypercube Sampling (LHS).

The key figure in this experiment is the number of evaluations n_{eval} which describes the number of expensive evaluations of the original function. During the experiment the ALS and the GA approach were stopped in case it was obvious that they will not converge.

For the genetic algorithm the number of evaluations n_{eval} is calculated as $n_{eval} = \mathit{population\_size} \times \mathit{number\_of\_generations}$. So, for instance, when the algorithm runs with a population size of 64 for 100 generations the algorithm runs for a maximum number of evaluations of $n_{eval} = 6,400$.

The active learning scheme only evaluates the original function during the validation of promising candidates suggested by the surrogate model. The top 10% of candidates, specifically those with the lowest objectives, are selected for further validation in each iteration. This percentage is defined by a parameter we termed "learning rate" lr. This value was determined in a set of test runs to find a low learning rate that yields overall good results. Thus, the number of evaluations in each iteration of the ALS depends on the sample size n and is calculated as $n_{eval} = \lfloor lr \times \mathit{population\ size} \rfloor$, i.e.

population size $= 256 \longrightarrow n_{eval} = 25$ per iteration. Function evaluations for the initial dataset are not included in n_{eval}.

7.1. Simple Quadratic (2-1)

This section describes the experimental results for the Simple Quadratic single-objective optimization problem with $k_{in} = 2$ input variables. This section presents the result table for the number of evaluation n_{eval} (see Table 7.1) and the number of generations / iterations (see Table 7.2). Also the convergence plots for sample size 64 (see Figure 7.1a) and for sample size 256 (see Figure 7.1b) are displayed. In order to visualize the content of the tables there are bar charts in Figure 7.2.

The ALS approach took far less evaluations of the original function than the genetic algorithm. Hence, the ALS converged much faster than the GA when solving the Simple Quadratic (2-1) problem up to an absolute error in the objective space of $\epsilon = 0.05$. Interestingly, the pseudo-random sampling (PRS) approach in the active learning scheme was the slowest ALS configuration for both sample size, but still it took less evaluations than the GA. In one instance the genetic algorithm was able to converge after a single generation and similarly the ALS could converge in a single iteration once as well. The small sample size $n = 64$ was large enough to provide sufficient information for convergence and so both solving approaches required less evaluations than with the large sample size.

Table 7.1.: Number of evaluations to solve Simple Quadratic (2-1)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
GS	64	24	768	25
PRS	128	36	768	325
SS	256	18	768	75
LHS	192	24	512	100

This table lists the number of evaluations n_{eval} to solve Simple Quadratic (2-1) with each configuration.

Table 7.2.: Number of generations/iterations to solve Simple Quadratic (2-1)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
GS	1	4	3	1
PRS	2	6	3	13
SS	4	3	3	3
LHS	3	4	2	4

This table lists the number of generations / iterations to solve Simple Quadratic (2-1) with each configuration.

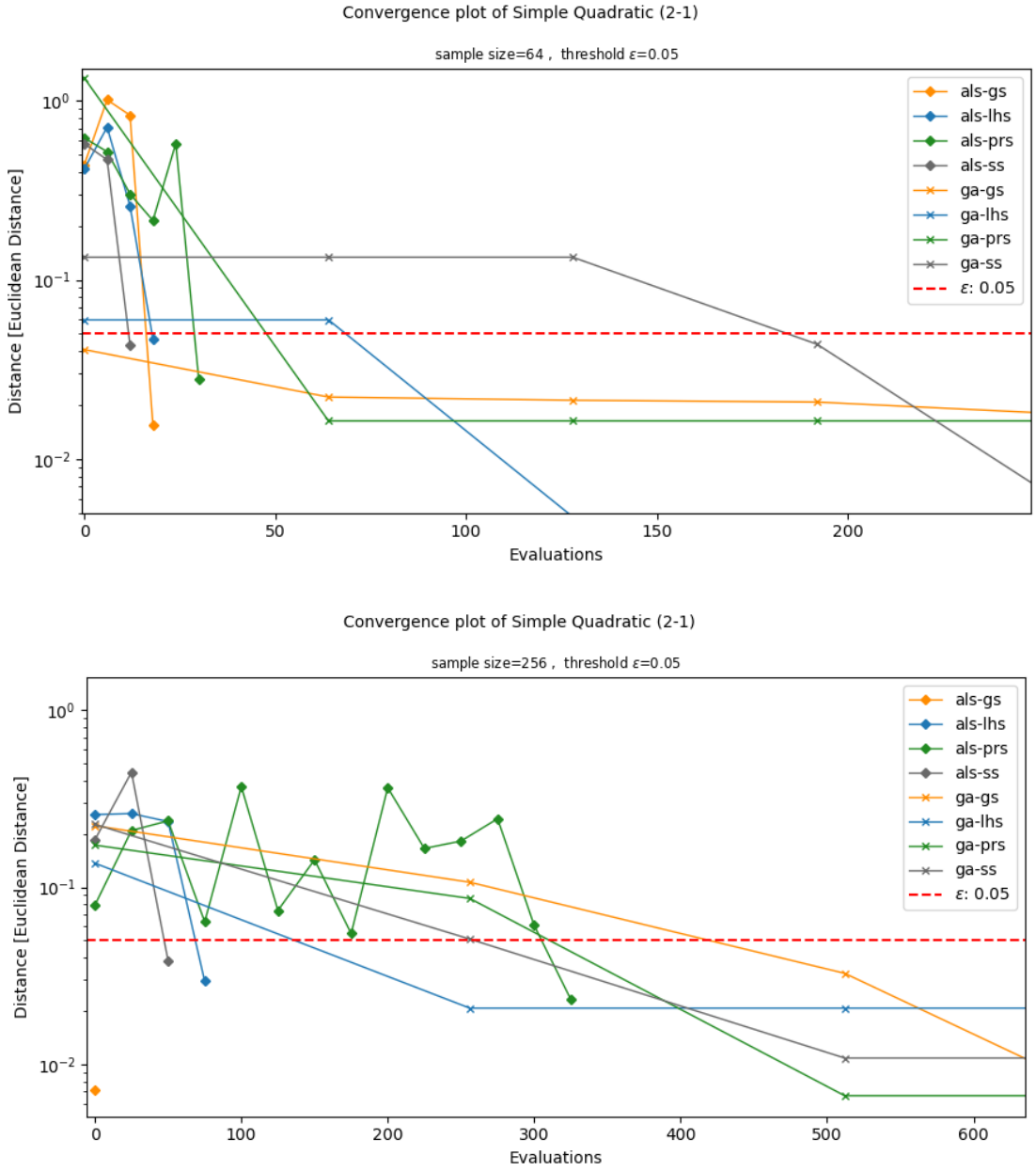


Figure 7.1.: Convergence plot of Simple Quadratic (2-1) problem with sample size $n = 64$ in the upper plot and $n = 256$ below. Colors indicate the sampling method. The lines marked by a cross shape mark the GA's generations and the diamond shapes indicate ALS iterations. On the X-axis there is the number of evaluations. The Y-axis measures the Euclidean distance to the global minimum in absolute terms on a log-scale.

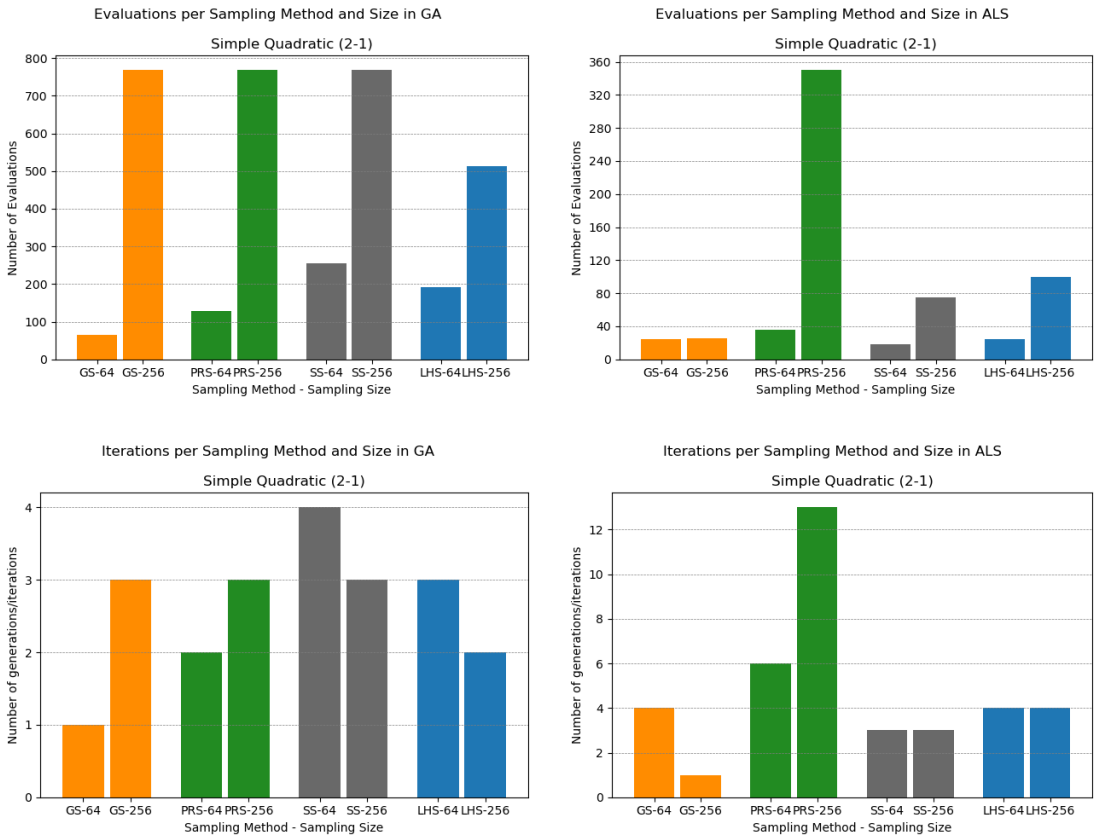


Figure 7.2.: Bar chart plots visualizing computational effort to solve Simple Quadratic (2-1) problem in any configuration. In the left column of this figure we see the numbers of the genetic algorithm (GA) approach and in the right column we see the results of the approach using the active learning scheme (ALS). The number of evaluations n_{eval} is depicted in the upper row and the number of iterations in the bottom row. Colors indicate the sampling method as well as the description on the X-axis.

7.2. Simple Quadratic (10-1)

This section describes the experimental results for the Simple Quadratic single-objective optimization problem with $k_{in} = 10$ input variables. This section presents the result table for the number of evaluation n_{eval} (see Table 7.3) and the number of generations / iterations (see Table 7.4). Also the convergence plots for sample size 64 (see Figure 7.3a) and for sample size 256 (see Figure 7.3b) are displayed. In order to visualize the content of the tables there are bar charts in Figure 7.4.

Grid sampling is not feasible in this experiment, because there are 10 input dimensions. Hence, the sample size would have to be very high to construct the grid and properly sample in each dimension. On the log-log convergence plot in Figure 7.3 we can observe that each configuration had a similar convergence behavior in the respective scheme while solving the Simple Quadratic problem with 10 variables. The Table 7.3 shows that the number of evaluations used in the genetic algorithm was about four times as high than in the ALS for sample size $n = 64$. This ratio was even higher for $n = 256$, where the GA required about five to six times more evaluations. Again, the small sample size was large enough and so both solving approaches required less evaluations than with the large sample size.

Table 7.3.: Number of evaluations to solve Simple Quadratic (10-1)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
PRS	2,176	528	7,168	1,425
SS	1,984	498	7,168	1,350
LHS	2,112	540	6,144	1,075

This table lists the number of evaluations n_{eval} to solve Simple Quadratic (10-1) with each configuration.

Table 7.4.: Number of generations/iterations to solve Simple Quadratic (10-1)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
PRS	34	88	28	57
SS	31	83	28	54
LHS	33	90	24	43

This table lists the number of generations / iterations to solve Simple Quadratic (10-1) with each configuration.

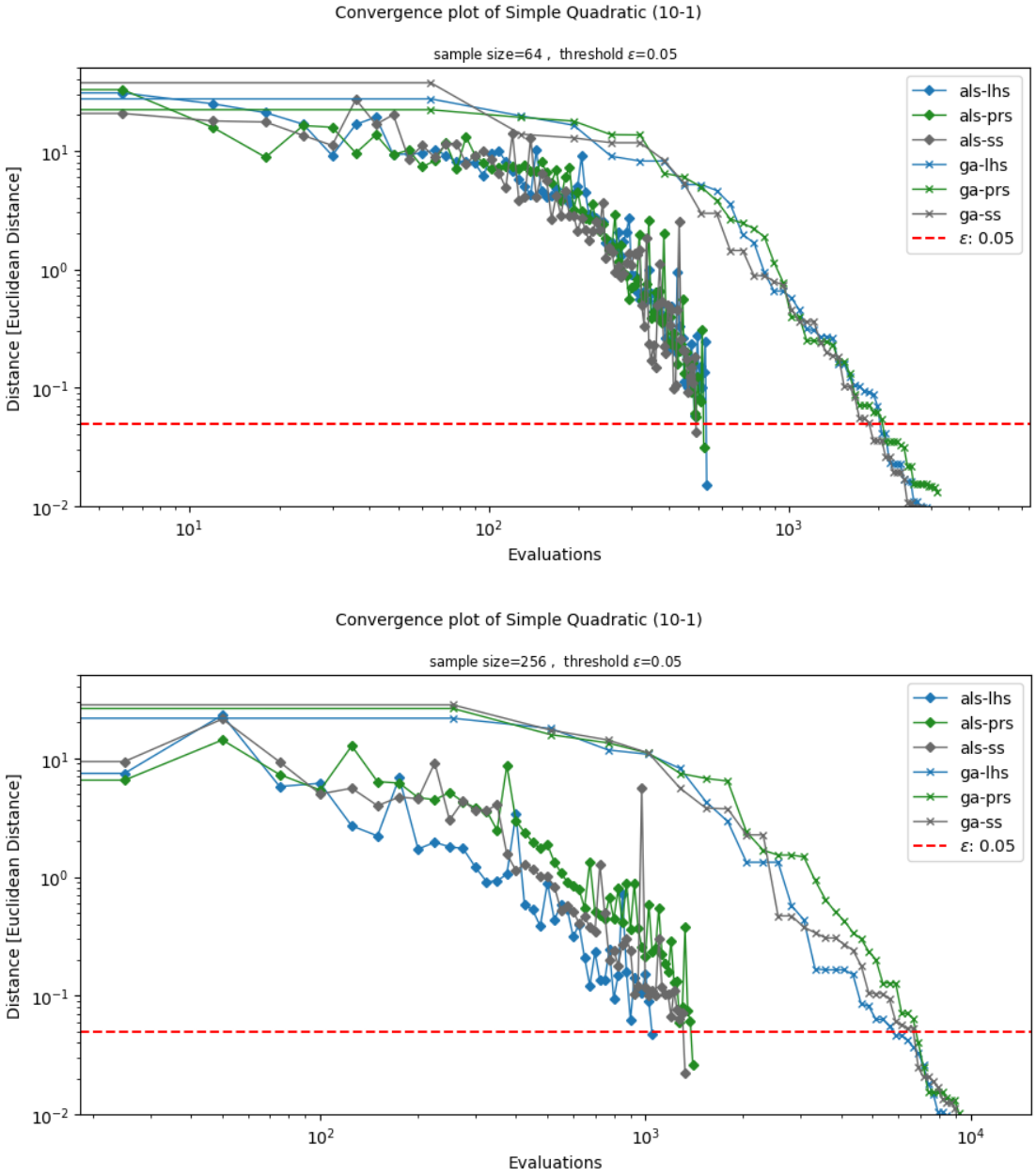


Figure 7.3.: Log-log convergence plot of Simple Quadratic (10-1) problem with sample size $n = 64$ in the upper plot and $n = 256$ below. Colors indicate the sampling method. The lines marked by a cross shape mark the GA's generations and the diamond shapes indicate ALS iterations. On the X-axis there is the number of evaluations. The Y-axis measures the Euclidean distance to the global minimum in absolute terms.

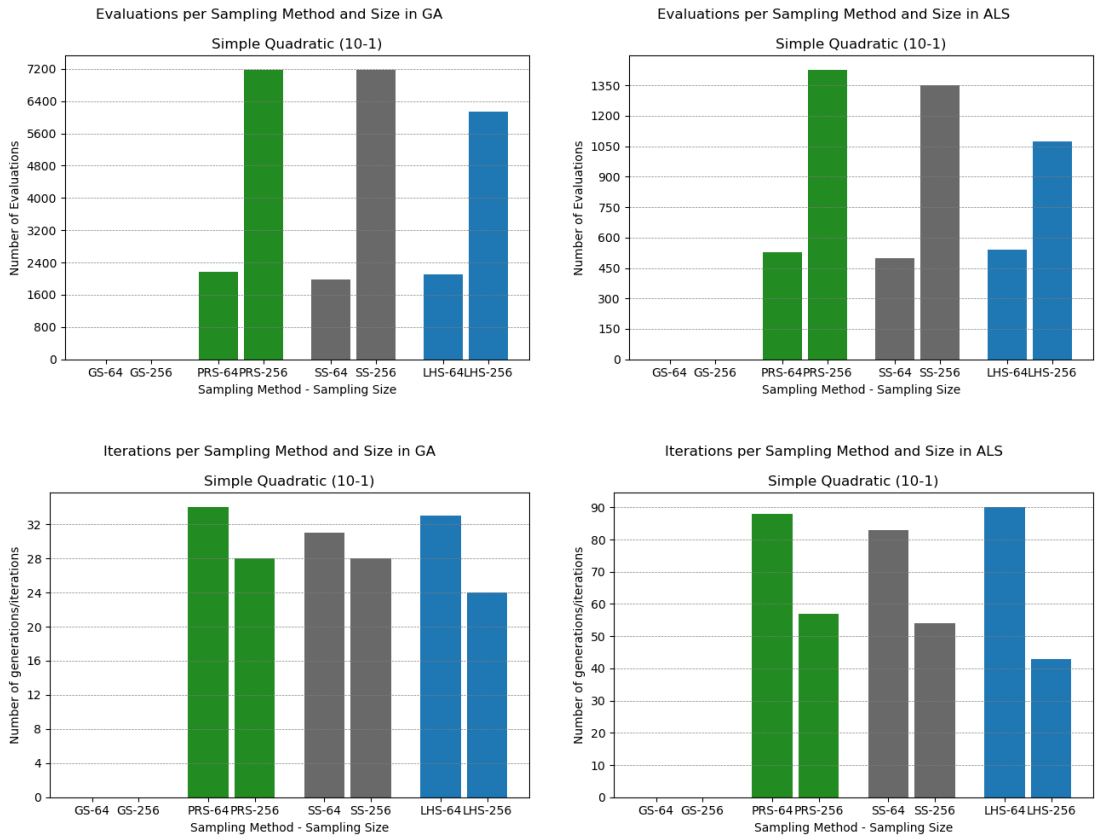


Figure 7.4.: Bar chart plots visualizing computational effort to solve Simple Quadratic (10-1) problem in any configuration. In the left column of this figure we see the numbers of the genetic algorithm (GA) approach and in the right column we see the results of the approach using the active learning scheme (ALS). The number of evaluations n_{eval} is depicted in the upper row and the number of iterations in the bottom row. Colors indicate the sampling method as well as the description on the X-axis.

7.3. Griewank (2-1)

This section describes the experimental results for the single-objective optimization test function Griewank. This section presents the result table for the number of evaluation n_{eval} (see Table 7.5) and the number of generations / iterations (see Table 7.6). Also the convergence plots for sample size 64 (see Figure 7.5a) and for sample size 256 (see Figure 7.5b) are displayed. In order to visualize the content of the tables there are bar charts in Figure 7.6.

The noisy Griewank (2-1) problem was not always solved by the ALS. Only for the small sample size $n = 64$ and the when we employed grid sampling (GS) or pseudo random sampling (PRS) the scheme was able to converge. In the experiments with the large sample size ALS did not converge at all, but only fluctuates a lot. The genetic algorithm had no problem with finding a solution within the error range and converged in under 1,500 evaluations with $n = 64$ and with about 1,500 to 3,000 evaluations with the large sample size $n = 256$. Again, the small sample size carried enough information for the GA to converge with less evaluations.

Table 7.5.: Number of evaluations to solve Griewank (2-1)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
GS	1,216	66	3,072	-
PRS	1,408	270	1,536	-
SS	640	-	2,560	-
LHS	1,216	-	3,072	-

This table lists the number of evaluations n_{eval} to solve Griewank (2-1) with each configuration.

Table 7.6.: Number of generations/iterations to solve Griewank (2-1)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
GS	19	11	12	-
PRS	22	45	6	-
SS	10	-	10	-
LHS	19	-	12	-

This table lists the number of generations / iterations to solve Griewank (2-1) with each configuration.

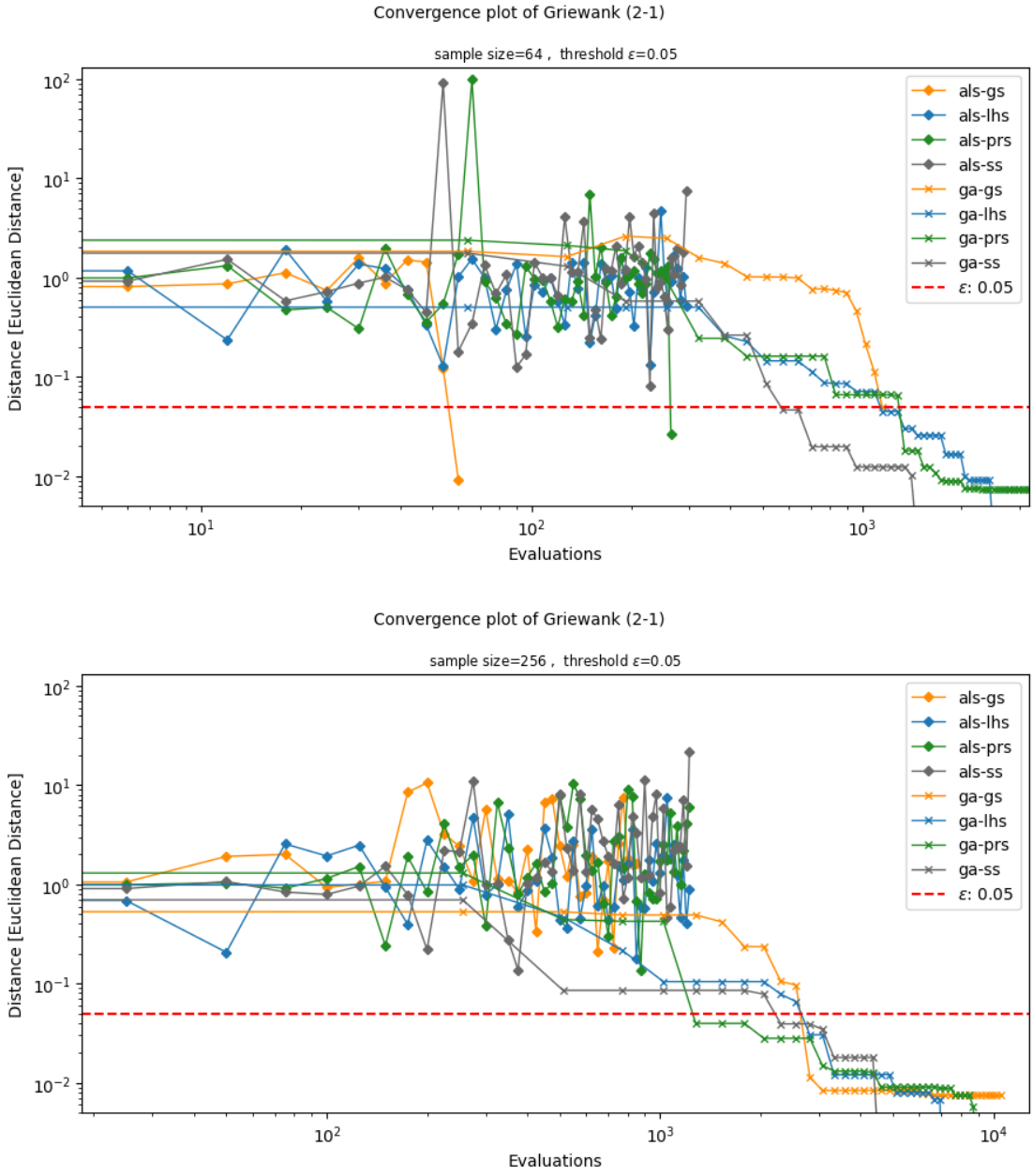


Figure 7.5.: Log-log convergence plot of Griewank (2-1) problem with sample size $n = 64$ in the upper plot and $n = 256$ below. Colors indicate the sampling method. The lines marked by a cross shape mark the GA's generations and the diamond shapes indicate ALS iterations. On the X-axis there is the number of evaluations. The Y-axis measures the Euclidean distance to the global minimum in absolute terms.

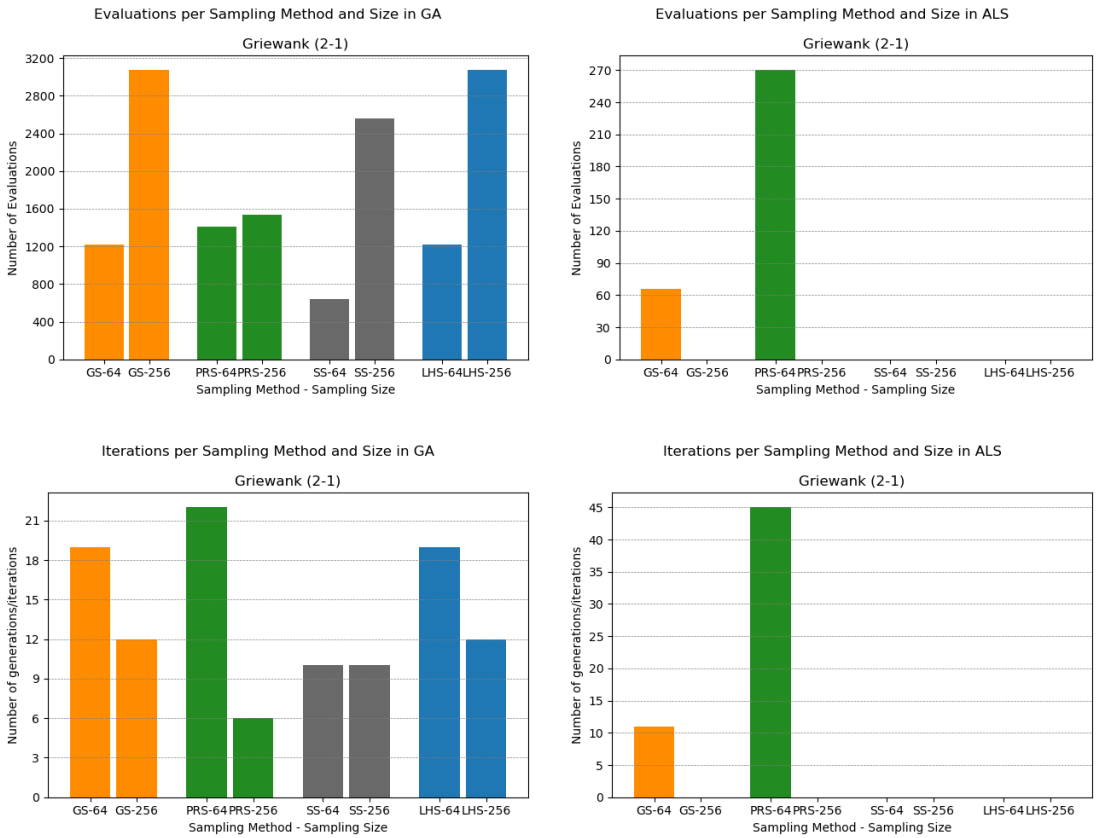


Figure 7.6.: Bar chart plots visualizing computational effort to solve Griewank (2-1) problem in any configuration. In the left column of this figure we see the numbers of the genetic algorithm (GA) approach and in the right column we see the results of the approach using the active learning scheme (ALS). The number of evaluations n_{eval} is depicted in the upper row and the number of iterations in the bottom row. Colors indicate the sampling method as well as the description on the X-axis.

7.4. Ackley (2-1)

This section describes the experimental results for the single-objective optimization test function Ackley. This section presents the result table for the number of evaluation n_{eval} (see Table 7.7) and the number of generations / iterations (see Table 7.8). Also the convergence plots for sample size 64 (see Figure 7.7a) and for sample size 256 (see Figure 7.7b) are displayed. In order to visualize the content of the tables there are bar charts in Figure 7.8.

The ALS converged much faster than the GA on this problem, despite the fact that the Ackley (2-1) function is fairly noisy. On sample size $n = 64$ the ALS converged about 15 - 56 times faster than the GA. This difference was less pronounced for $n = 256$, nevertheless the ALS was about 4 - 15 times faster than the GA. Again, the small sample size enabled both solving approaches to converge with less evaluations than with the large sample size.

Table 7.7.: Number of evaluations to solve Ackley (2-1)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
GS	1,664	54	3,584	500
PRS	1,856	120	4,352	350
SS	2,368	42	4,096	275
LHS	1,472	90	2,084	475

This table lists the number of evaluations n_{eval} to solve Ackley (2-1) with each configuration.

Table 7.8.: Number of generations/iterations to solve Ackley (2-1)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
GS	26	9	14	20
PRS	29	20	17	14
SS	37	7	16	11
LHS	23	15	8	19

This table lists the number of generations / iterations to solve Ackley (2-1) with each configuration.

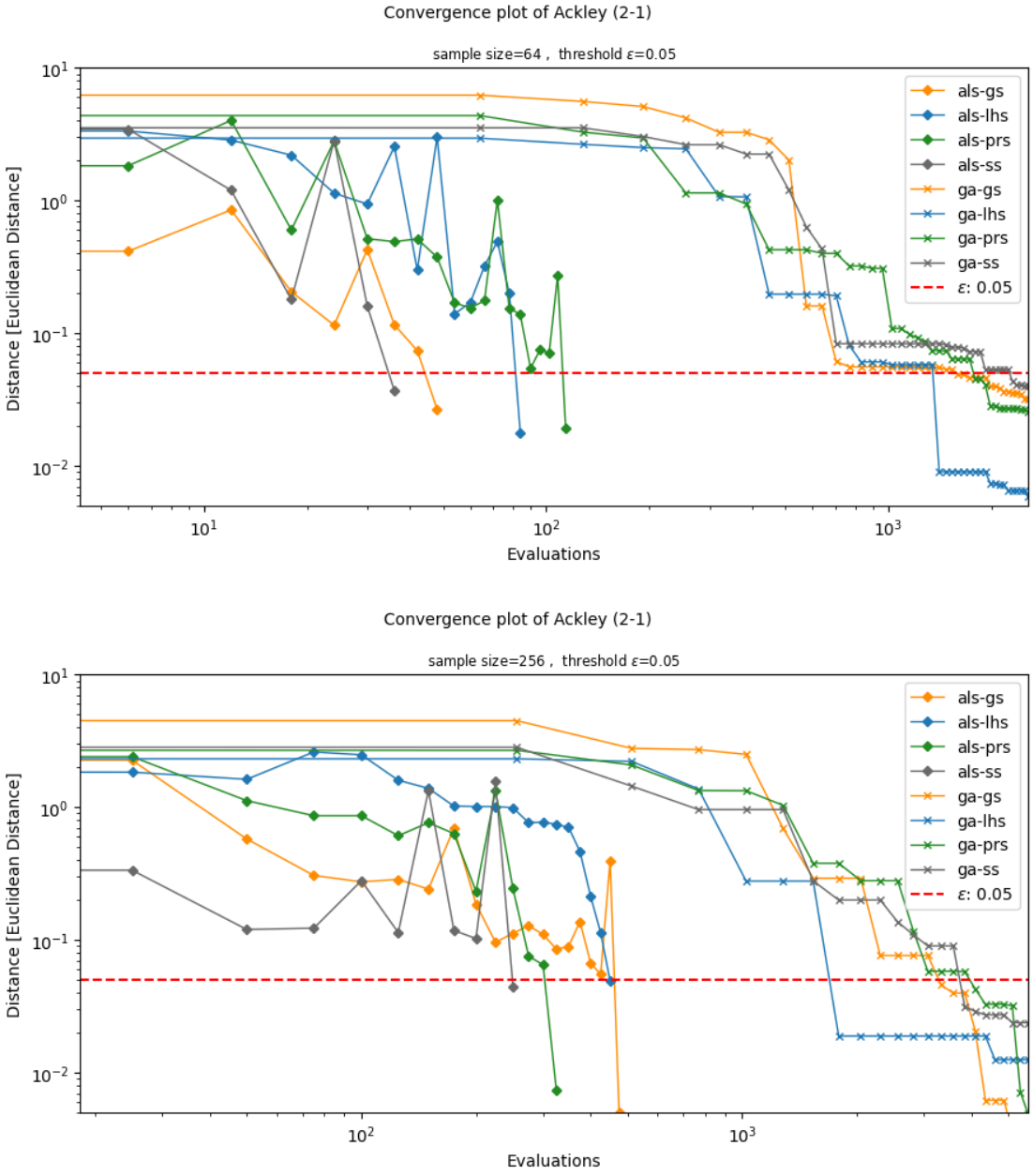


Figure 7.7.: Log-log convergence plot of Ackley (2-1) problem with sample size $n = 64$ in the upper plot and $n = 256$ below. Colors indicate the sampling method. The lines marked by a cross shape mark the GA's generations and the diamond shapes indicate ALS iterations. On the X-axis there is the number of evaluations. The Y-axis measures the Euclidean distance to the global minimum in absolute terms.

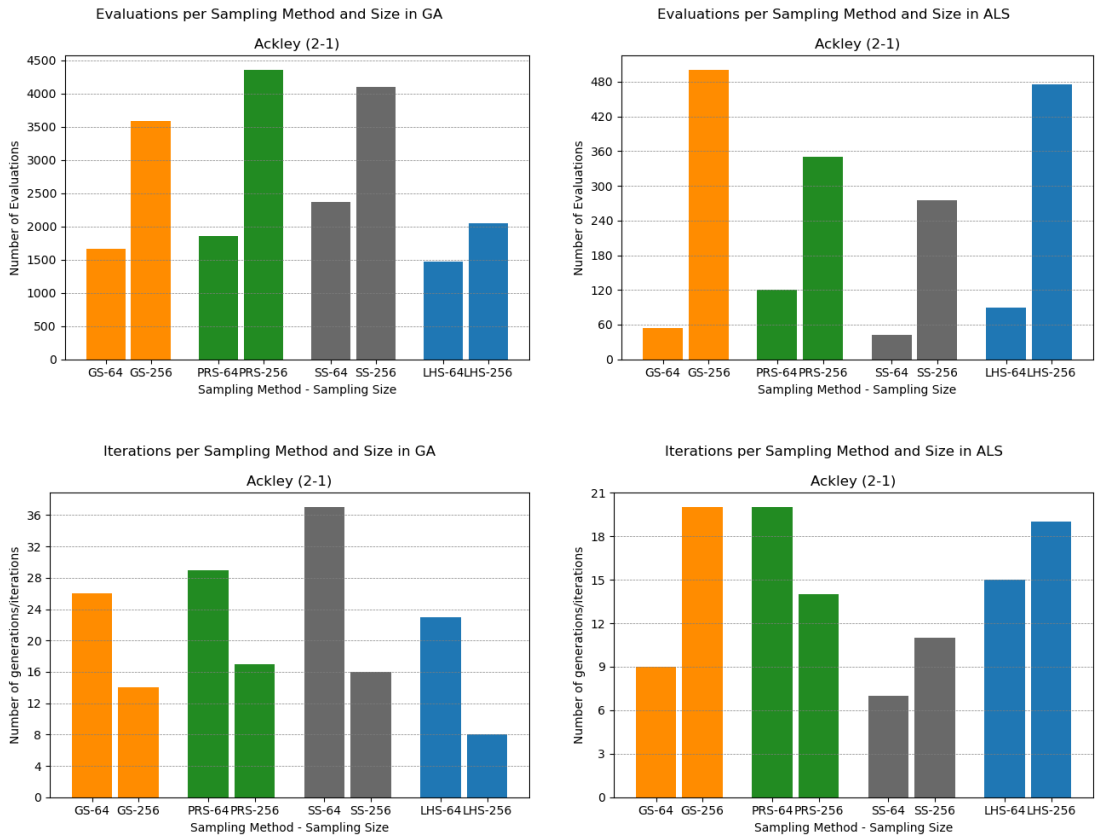


Figure 7.8.: Bar chart plots visualizing computational effort to solve Ackley (2-1) problem in any configuration. In the left column of this figure we see the numbers of the genetic algorithm (GA) approach and in the right column we see the results of the approach using the active learning scheme (ALS). The number of evaluations n_{eval} is depicted in the upper row and the number of iterations in the bottom row. Colors indicate the sampling method as well as the description on the X-axis.

7.5. ZDT1 (2-2)

This section describes the experimental results for the multi-objective optimization test function ZDT1 in 2-dimensional decision space. This test function has a smooth surface. This section presents the result table for the number of evaluation n_{eval} (see Table 7.9) and the number of generations / iterations (see Table 7.10). Also the convergence plots for sample size 64 (see Figure 7.9a) and for sample size 256 (see Figure 7.9b) are displayed. In order to visualize the content of the tables there are bar charts in Figure 7.10 however the bar charts for the ALS were left out as there are no notable results. Lastly, in Figure 7.11 all datapoints that were considered by the ALS during the optimization of this problem on sample size $n = 64$ are plotted.

The genetic algorithm solved the ZDT1 (2-2) problem without any problems. Interestingly the GA converged clearly the fastest when using grid sampling to generate its populations.

The active learning scheme was not able to meet the defined convergence threshold of $\epsilon = 0.05$ measured by means of the IGD for this multi-objective problem. We can see in Figure 7.9 that the IGD remained on almost the same exact level of around 0.3 on all sample approaches in the ALS's results.

Table 7.9.: Number of evaluations to solve ZDT1 (2-2)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
GS	128	-	256	-
PRS	448	-	1,024	-
SS	384	-	768	-
LHS	320	-	768	-

This table lists the number of evaluations n_{eval} to solve ZDT1 (2-2) with each configuration.

Table 7.10.: Number of generations/iterations to solve ZDT1 (2-2)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
GS	2	-	1	-
PRS	7	-	4	-
SS	6	-	3	-
LHS	5	-	3	-

This table lists the number of generations / iterations to solve ZDT1 (2-2) with each configuration.

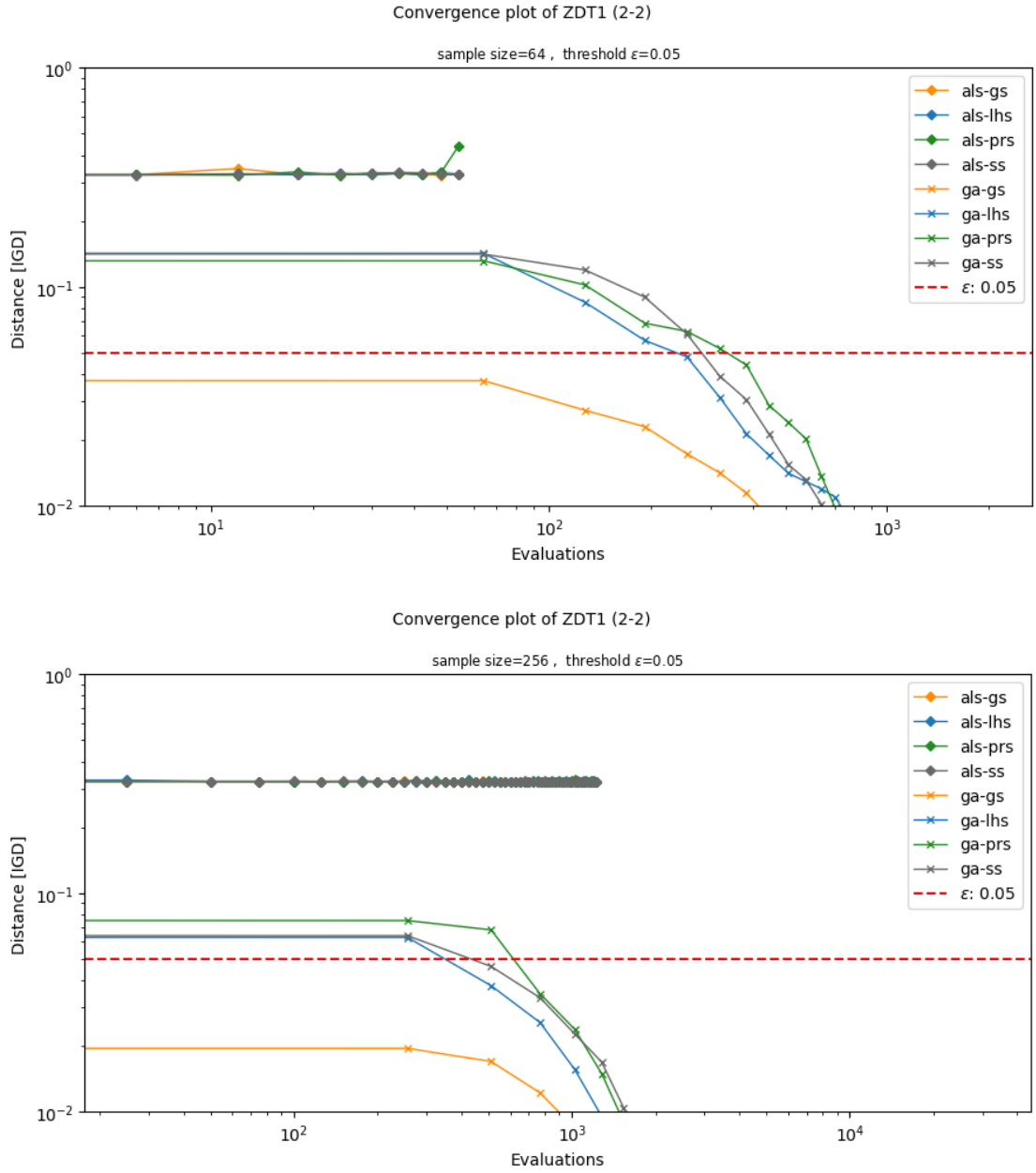


Figure 7.9.: Log-log convergence plot of ZDT1 (2-2) problem with sample size $n = 64$ in the upper plot and $n = 256$ below. Colors indicate the sampling method. The lines marked by a cross shape mark the GA's generations and the diamond shapes indicate ALS iterations. On the X-axis there is the number of evaluations. The Y-axis measures the IGD.

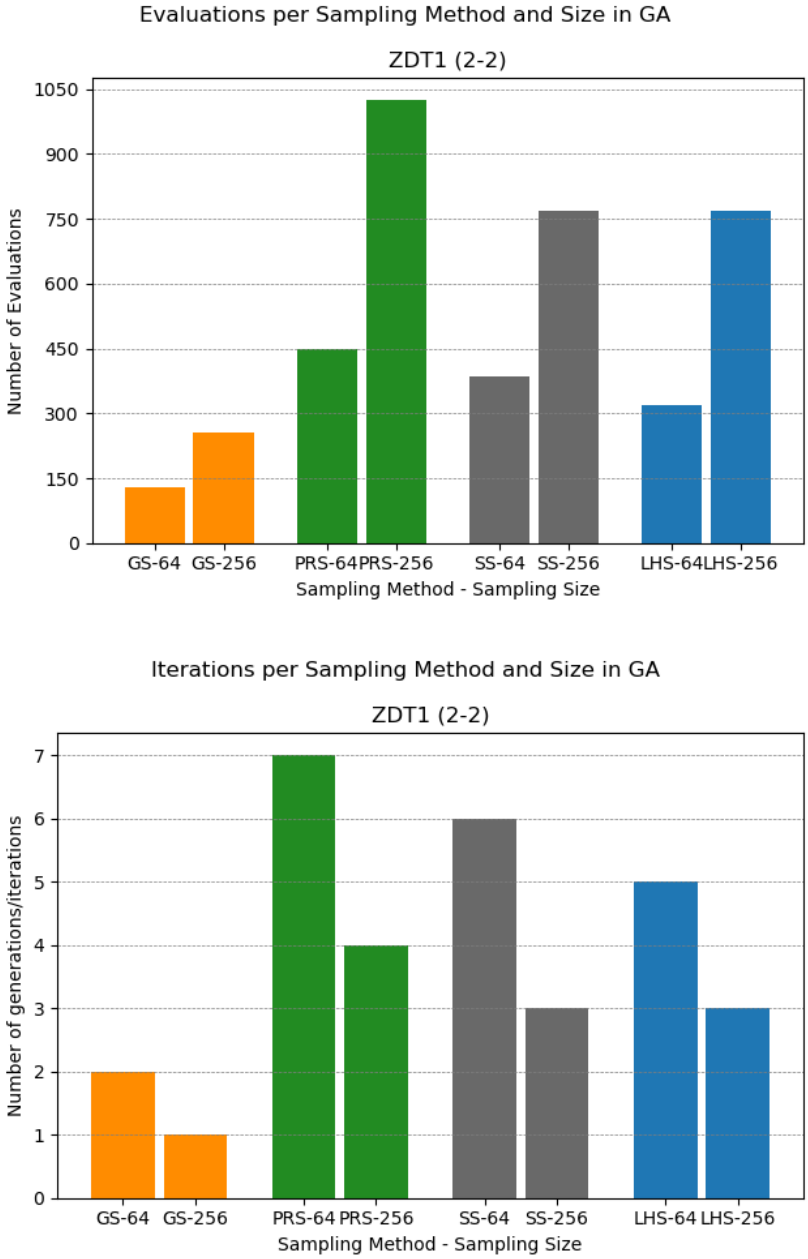
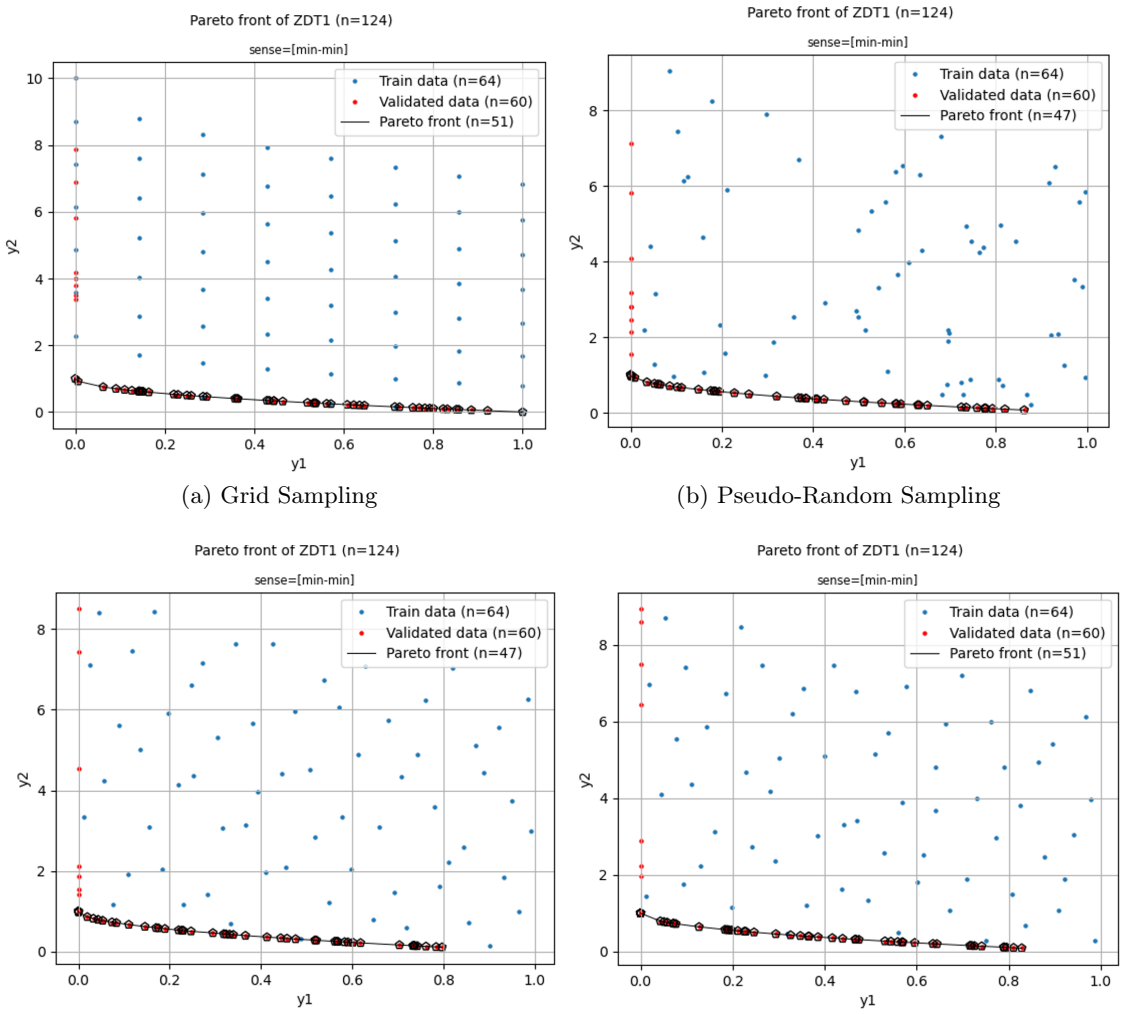


Figure 7.10.: Bar chart plots visualizing computational effort to solve ZDT1 (2-1) problem in any configuration, solved with the GA. The number of evaluations n_{eval} is depicted in the upper row and the number of iterations in the bottom row. Colors indicate the sampling method as well as the description on the X-axis.



(a) Grid Sampling

(b) Pseudo-Random Sampling

(c) Sobol' Sequence

(d) Latin Hypercube Sampling

Figure 7.11.: This figure shows 4 plots visualizing the datapoints considered by the ALS with the respective sampling method indicated by the caption beneath each plot. These plots show the initial train dataset in blue and the validated data points in red in the 2-dimensional objective space. In this example, the scheme solved the ZDT1 (2-2) problem with a sample size of $n = 64$. The ALS evaluated 60 data points and most of them are located on the true Pareto front. The true Pareto is given as the black line with diamonds, where each diamond contains a validated point.

Let us now take a look at Figure 7.11 which shows four plots of the active learning scheme's data in the objective space. The initial dataset of $n = 64$ points is colored blue and the validated data of 60 points found by the ALS is colored red. The true Pareto is given as the black line with diamonds, where each diamond contains a validated point.

When we examine the location of the red validated points, it becomes evident that the scheme found a lot of solutions along the true Pareto front. The exact number of Pareto efficient solutions is given in the legend in each plot ("*Pareto front (n=..)*") and lies between 47 and 51 in this case.

So, while the IGD metric did not determine that the scheme converged successfully, it found nevertheless 47 to 51 solutions at the expense of $n_{eval} = 60$ evaluations in total. What is also interesting about Figure 7.11 is that the Pareto front is best explored in the interval $[0, 1]$ on the y_1-axis when the ALS had an initial dataset generated with grid sampling. This result suggests that the grid sampling improved the schemes exploration of the solution space.

7.6. ZDT1 (10-2)

This section describes the experimental results for the multi-objective optimization test function ZDT1 in 10-dimensional decision space. This test function has a smooth surface, however the high dimensionality makes the problem harder to solve. This section presents the result table for the number of evaluation n_{eval} (see Table 7.11) and the number of generations / iterations (see Table 7.12). Also the convergence plots for sample size 64 (see Figure 7.12b) and for sample size 256 (see Figure 7.12a) are displayed. In order to visualize the content of the tables there are bar charts in Figure 7.10 however the bar charts for the ALS were left out as there are notable results. Lastly, in Figure 7.14 all datapoints that were considered by the ALS during the optimization of this problem on sample size $n = 64$ are plotted.

Grid sampling is not feasible in this experiment, because there are 10 input dimensions. Hence, the sample size would have to be very high to construct the grid and properly sample in each dimension. The active learning scheme was not able to converge on this multi-objective problem. We can see in Figure 7.12 that the convergence on all sample approaches in the ALS remained on almost the same exact level, where the IGD measured a little more than 0.3. The same behaviour was observed when solving the multi-objective function ZDT1 with 2 input variables. The genetic algorithm solved the ZDT1 (10-2) problem without any problems. The number evaluations was almost the same on all three sampling methods with respect to the sample size as seen Figure 7.13.

Table 7.11.: Number of evaluations to solve ZDT1 (10-2)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
PRS	1,728	-	4,608	-
SS	1,792	-	4,864	-
LHS	1,536	-	5,120	-

This table lists the number of evaluations n_{eval} to solve ZDT1 (10-2) with each configuration.

Table 7.12.: Number of generations/iterations to solve ZDT1 (10-2)

	64 GA	**64 ALS**	**256 GA**	**256 ALS**
PRS	27	-	18	-
SS	28	-	19	-
LHS	24	-	20	-

This table lists the number of generations / iterations to solve ZDT1 (10-2) with each configuration.

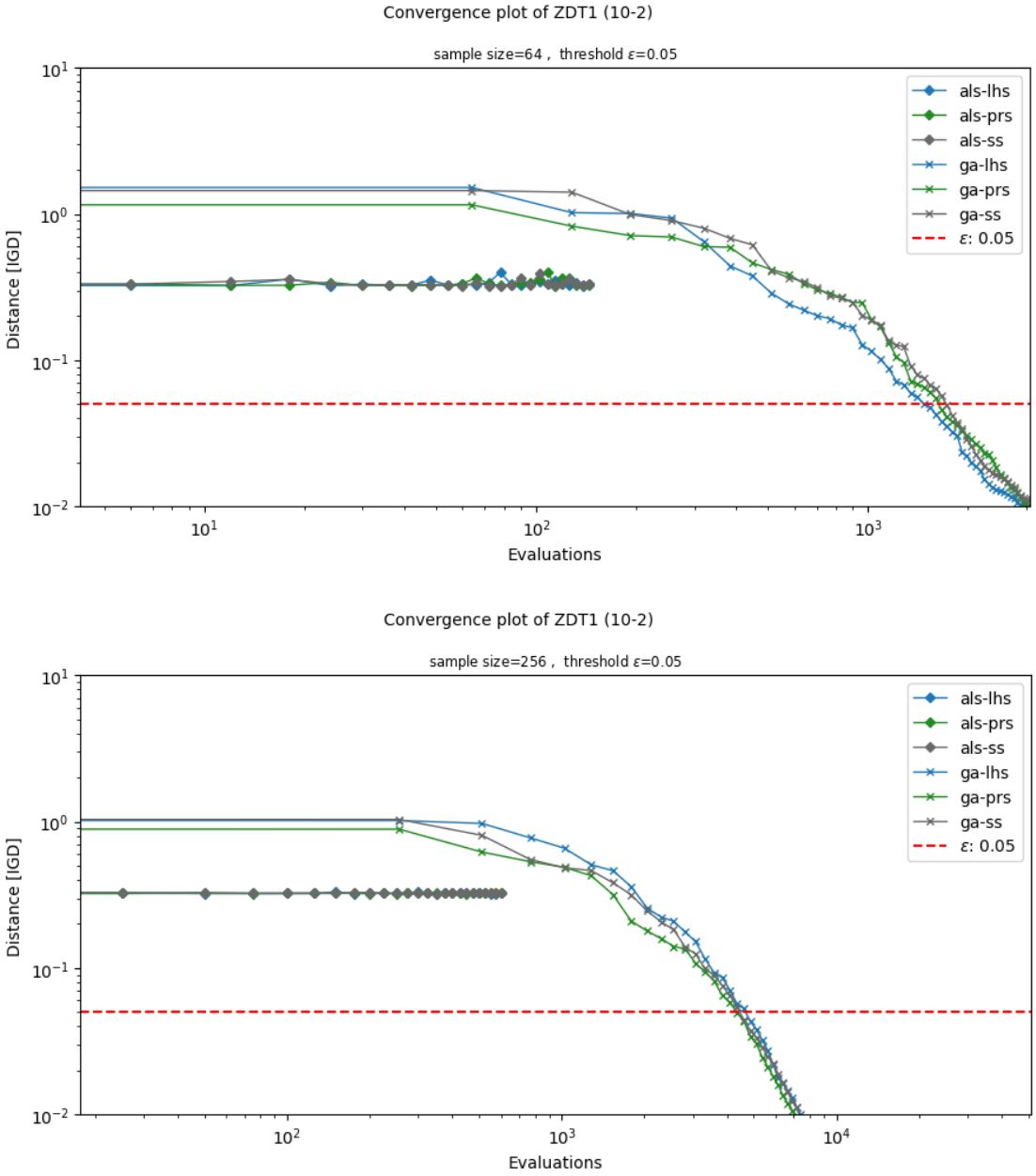


Figure 7.12.: Log-log convergence plot of ZDT1 (10-2) problem with sample size $n = 64$ in the upper plot and $n = 256$ below. Colors indicate the sampling method. The lines marked by a cross shape mark the GA's generations and the diamond shapes indicate ALS iterations. On the X-axis there is the number of evaluations. The Y-axis measures the IGD.

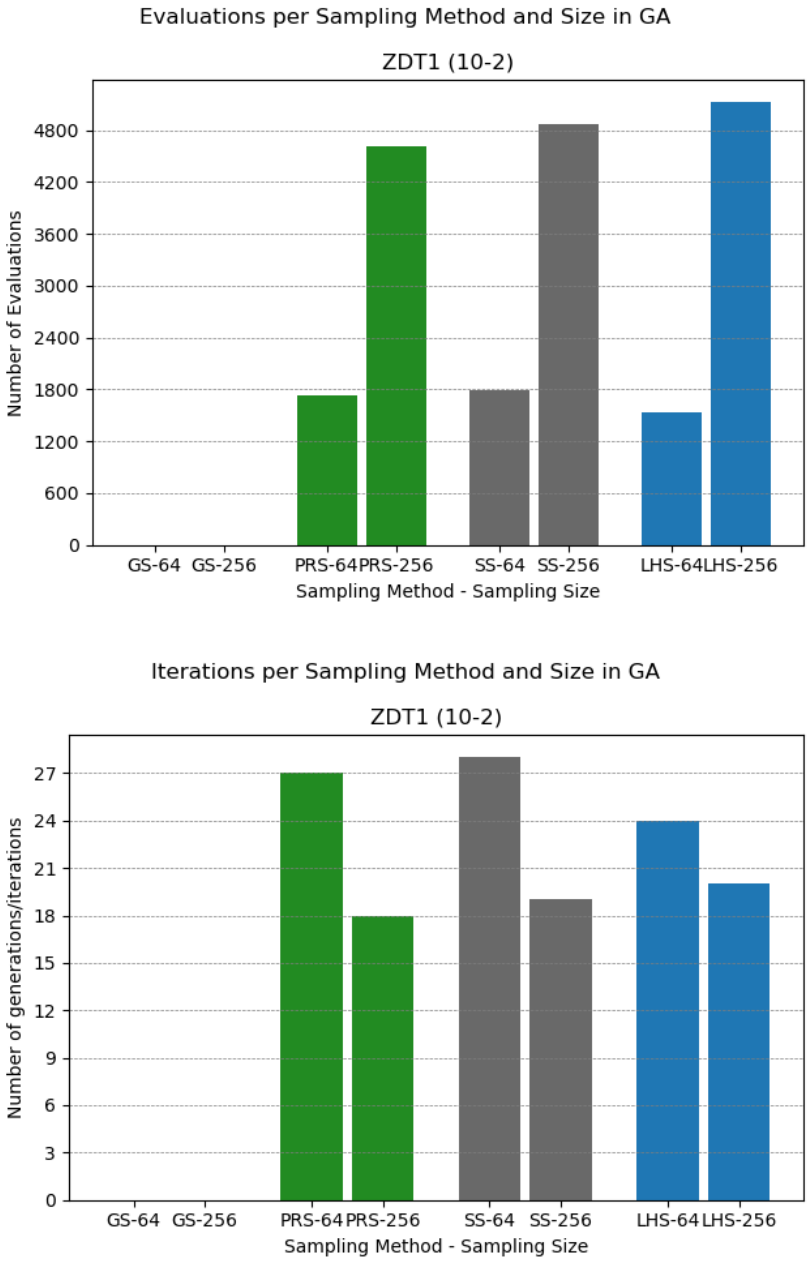
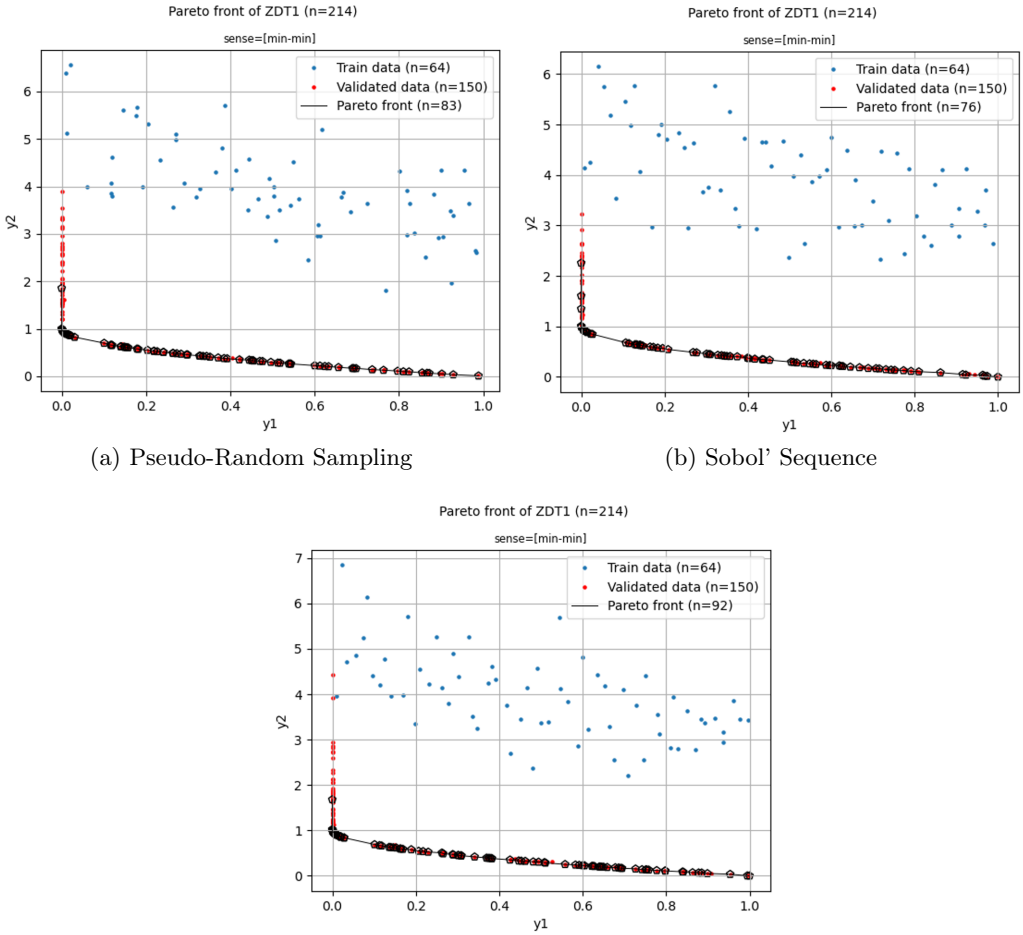


Figure 7.13.: Bar chart plots visualizing computational effort to solve ZDT1 (10-1) problem in any configuration, solved with the GA. The number of evaluations n_{eval} is depicted in the upper row and the number of iterations in the bottom row. Colors indicate the sampling method as well as the description on the X-axis.



(a) Pseudo-Random Sampling

(b) Sobol' Sequence

(c) Latin Hypercube Sampling

Figure 7.14.: This figure shows 4 plots visualizing the datapoints considered by the ALS with the respective sampling method indicated by the caption beneath each plot. These plots show the initial train dataset in blue and the validated data points in red in the 2-dimensional objective space. In this example, the scheme solved the ZDT1 (10-2) problem with a sample size of $n = 64$. The ALS evaluated 150 data points and most of them are located on the true Pareto front. The true Pareto is given as the black line with diamonds, where each diamond contains a validated point.

If we examine Figure 7.14 we make a similar observation as in the previous experiment. We find that the ALS is able to find plenty of solution located on the true Pareto front marked with a diamond even with a small initial dataset of $n = 64$ points. Here, after the ALS evaluated 150 points it found 76 to 92 Pareto optimal solutions. However, if we compare Figure 7.11 from earlier and Figure 7.14 then we observe that here in ZDT (10-2) the scheme validated even more "stray" points along the y_2-axis at $y_1 = 0$.

8. Discussion and Outlook

This chapter discusses the results obtained during the parameter study. Advantages and shortcomings of the current implementation of the active learning scheme are reviewed. Additionally, an outlook on what can be improved is given.

Limited Exploration and Stray Points in ALS for Multi-Objective Optimization

A specific issue that was observed were "stray" points also seen in Figure 7.11 & 7.14 which gather along the boundary of the y_2-axis where $y_1 = 0$ located far from the Pareto front. In the higher dimensional problem ZDT1 where $k_{in} = 10$ there were more stray points than in the lower dimensional case where $k_{in} = 2$. This issue is especially visible when comparing the ALS's solutions with the population development of the genetic optimization algorithm. To compare the active learning scheme's solutions with the genetic algorithm's solutions we take a look at Figure 8.1a. The red points in the upper plot of this figure offer no value for the IGD metric. In contrast, when we look at the lower plot of Figure 8.1b, there the NSGA-II demonstrated superior performance in exploring the solution space, evidenced by its effective coverage along the Pareto front and finding less stray points in its final generations.

Unfortunately, the persistence of these stray points from the first iteration highlights the ALS's inefficiency in decision space exploration in its current implementation. This raises questions about whether the machine learning meta model was suitable, or whether the convergence criterion for multi-objective problems was defined correctly. The ALS limited capacity to properly explore the space most probably contributed to this issue.

Fine-Tuning of the Active Learning Scheme

During the experiment it became evident that there are two important parameters that can be fine-tuned: the sample size n and the learning rate lr. Here we have the potential to further improve the active learning scheme's efficiency. Minimizing the sample size allows us to use the ALS with a minimal amount of expensive initial data points. Also, we can lower the computational cost further, by carefully tuning the learning rate to proceed with less validations.

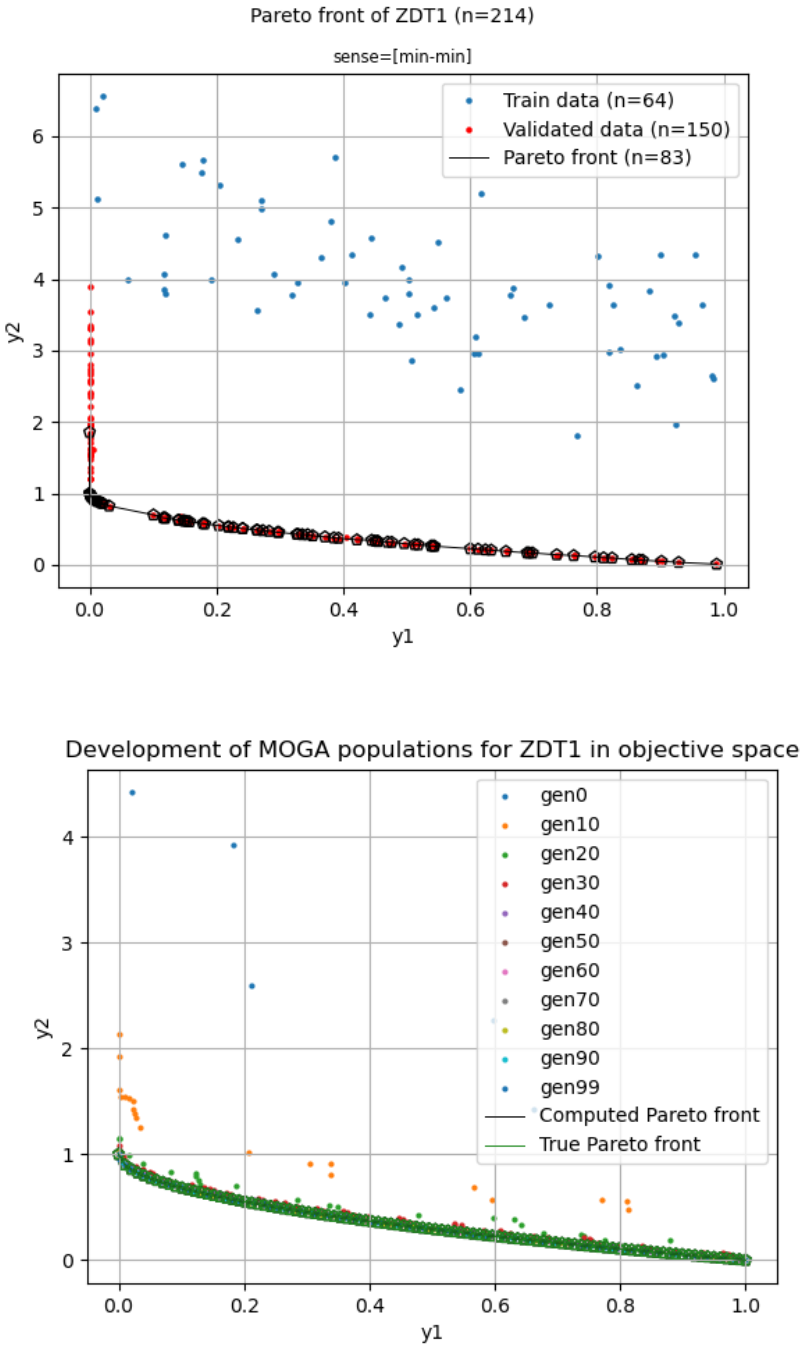


Figure 8.1.: This figure shows final population of the active learning scheme after 25 iterations in the upper plot. In the lower plot we can see the population of the genetic algorithm during a few selected generations from 0-99 total generations. This figure visualizes how the two optimization approaches approximate the true Pareto front of the ZDT1 (10-2) problem with $k_{in} = 10$ input variables and an initial sample size of $n = 64$.

Multi-Objective Function Challenges in Active Learning Scheme

The Active Learning Scheme (ALS) exhibited limitations in solving multi-objective functions (see results in Sections 7.5 & 7.6), most probably due to its lack of an exploration mechanism. This major shortcoming arises from the system's operational design:

1. The ALS does not systematically consider less explored areas, whether through a measure of uncertainty or through randomly sampling in regions close to the current optimum. If the scheme knows which areas are well explored and which predictions can be trusted, we might improve the domain exploration.

2. The ALS iteratively validates only the current minimal points, which inherently limits its capacity to explore new potential solutions.

3. There is a tendency for the ALS to validate the same solutions or ones that are in close proximity within the decision space multiple times, which leads to inefficient use of computational resources.

This limited exploration ability is particularly problematic in the context of noisy optimization landscapes, such as the Griewank problem as seen in Figure 8.2. This is why the ALS with its limited exploration was not able to converge on this problem in most cases.

However, in practical applications like material optimization, where optimization function landscapes are smoother than the test function used in this study, this lack of exploration would not have affected the optimum search as much.

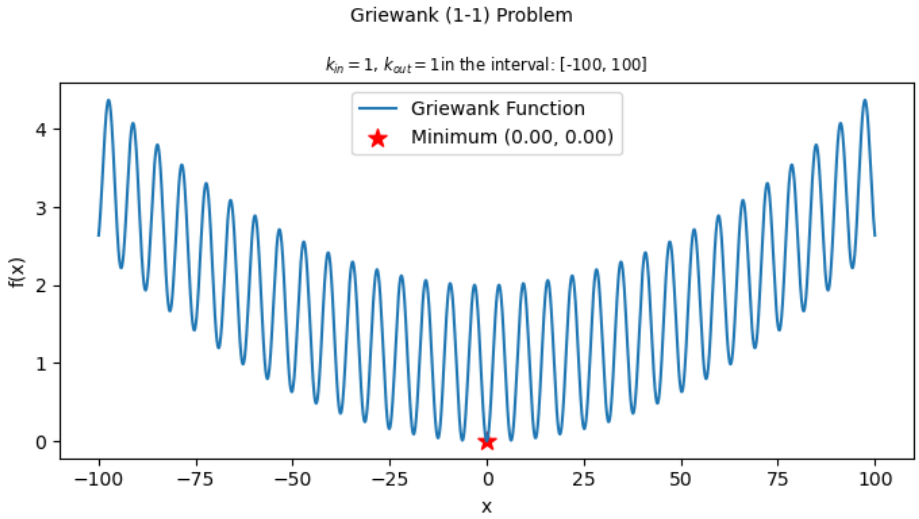


Figure 8.2.: Griewank (1-1) test function with $k_{in} = 1$ in interval $[-100, 100]$ around minimum. The objective function $f(x)$ is plotted against the variable x. This figure shows how noisy the region around the function's minimum is.

Outlook for Exploration Enhancement

To address the issues listed above, some potential strategies are proposed:

1. Using a *Gaussian Process Regressor* (i.e. implemented in scikit-learn [1] [31]) as surrogate model would be an option to obtain an uncertainty measure. These models provide a standard deviation value alongside their predictions. Based on their confidence level, suggested solutions can be excluded from the population or selected for evaluation to improve the surrogate model in subsequent iterations.

2. In order to avoid local extrema we could apply a variant of mini-batching during the training of the surrogate model. Mini-batching is applied in stochastic gradient descent to "tunnel through" local extrema and thus avoid getting stuck on sub-optimal solutions. In mini-batching essentially many machine learning models are trained on fractions of the full dataset. Thus, each model resembles a different response surface according to its training data. When combining their predictions, we obtain a response surface that fluctuates with respect to the main response surface (trained on the full dataset). Using these fluctuations might help the ALS to navigate through noisy functions [15, 14].

3. We want to prevent the validation of any new solution $\tilde{x}$ suggested by the meta model which is close to any already validated solution x in the database. Therefore, we could implement a mechanism that checks if any x in the database is in close proximity to a new $\tilde{x}$. For instance we could measure the Euclidean distance $d(\tilde{x}, x_i)$ in the decision space for all N solutions in the database in the range $i = (1, 2, ..., N)$. If this distance measure does not fall under a specific threshold for any known data point x, the new point $\tilde{x}$ will be validated to expand the information in the database.

Sampling Approach and Convergence Behavior

An intriguing observation was the minimal impact of the sampling method on the convergence behavior of the optimization algorithms. However, some effects have been observed:

1. In some cases when Pseudo Random Sampling (PRS) was involved the optimization approach exhibited a more erratic convergence behavior and slower convergence speed than approaches employing one of the other three sampling methods (GS, LHS, SS).

2. In higher input dimensions ($k_{in} = 10$), the differences in convergence behavior were diminished almost entirely. So, all three feasible sampling methods were equally effective in supporting both optimization approaches.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessRegressor.html

3. Grid Sampling (GS) was especially helpful for the ALS in exploring the Pareto front and covering its full length as seen in Figure 7.11a This structural sampling method generates a very evenly distributed dataset by design, which is able to inform the surrogate model well.

The first two observations held true for both the active learning scheme and the genetic algorithm. These findings suggest that the choice of the sampling method may not be very essential in these optimization contexts, but can play a beneficial role.

Although the sampling methods' advantages were not too obvious, considering the inherent benefits of Latin Hypercube sampling and sampling from the Sobol' sequence, they are considered to be good choices for sampling in high-dimensional space.

Evaluation Count and Efficiency in Optimization Tasks

In the context of optimization problems the ALS demonstrated the ability to reduce the number of potentially expensive function evaluations in the context of optimization compared to the genetic algorithm NSGA-II.

On **single-objective** test functions the ALS converged extremely fast on the Ackley function as described in Section 7.4 and also required far less function evaluations on the two variants of the Simple Quadratic problem (see Section 7.1 & 7.2). The Griewank function was too hard for the current implementation to reliably find a minimum that meets the convergence threshold (Section 7.3).

In the **multi-objective** case the active learning scheme attempted to solve the problem ZDT1 with 2 and with 10 variables. The ALS did not manage to converge by means of the Inverted Generational Distance (IGD) (Equation 2.3) on the multi-objective test function. The defined convergence criterion measures the spacial proximity of the computed Pareto front to the true Pareto front. By design the IGD requires an even coverage of the computed Pareto front to yield a good score. On the one hand, the active learning scheme in its current implementation could not compute the Pareto front with good coverage. On the other hand, when we take a look at the resulting Pareto fronts in Figure 7.11 & 7.14, we can observe that most of the scheme's solutions lie on the true Pareto front (black line), at the expense of comparably very few objective function evaluations.

Final Remarks and Disclaimer

It is crucial to note that the results presented in this study are not statistically significant due to the limited repetition of differently configured experiments. These results should be regarded as proof-of-concept, illustrating potential differences between approaches and revealing significant weaknesses in the current implementation of the active learning scheme for optimization tasks.

9. Conclusion

The study presented in this thesis aims to investigate the efficiency and effectiveness of the active learning scheme (ALS) in solving optimization problems. In a comparative parameter study the genetic algorithm NSGA-II serves as benchmark. Additionally, we investigate the influence of four different sampling methods on the solving efficiency, which turn out to have only minor effects.

Throughout the analysis, we identify various limitations and strengths our current implementation of the ALS, and potential improvements are proposed. The ALS, in its present iteration, is efficient in simple, more predictable environments with little noise but struggles in more complex scenarios due to its limited exploration capability.

In conclusion, the active learning scheme demonstrates notable efficiency in solving both single- and multi-objective problems, outperforming NSGA-II in scenarios involving simple to moderately complex problems, despite certain computational inefficiencies in our current implementation. However, in the context of highly noisy optimization tasks, NSGA-II reliably converges while the ALS faces difficulties overcoming local minima. Further, on multi-objective optimization problems the NSGA-II shows superior exploration capabilities of the Pareto front.

The comparative advantage of ALS over other optimization algorithms, particularly for simple problems where gradient-based methods may offer quicker and more reliable convergence behavior, remains an area for further investigation. Moreover, future research is needed to test the active learning scheme's applicability to other tasks, including different optimization problems, and to determine the necessary conditions to employ the ALS efficiently. The importance for more efficient solution strategies for complex optimization tasks is obvious, thus it will be of great interest how the ALS can be expanded and tailored to the respective applications. The potential strategies outlined in this study offer a roadmap for future research and development in this field.

Bibliography

[1] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer. [Online]. Available: https://doi.org/10.1007/0-387-22742-3_18

[2] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient Global Optimization of Expensive Black-Box Functions."

[3] A. Kovacs, J. Fischbacher, H. Oezelt, A. Kornell, Q. Ali, M. Gusenbauer, M. Yano, N. Sakuma, A. Kinoshita, T. Shoji, A. Kato, Y. Hong, S. Grenier, T. Devillers, N. M. Dempsey, T. Fukushima, H. Akai, N. Kawashima, T. Miyake, and T. Schrefl, "Physics-informed machine learning combining experiment and simulation for the design of neodymium-iron-boron permanent magnets with reduced critical-elements content," vol. 9, p. 1094055. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fmats.2022.1094055/full

[4] J. Li, T. Sasaki, T. Ohkubo, and K. Hono, "Most frequently asked questions about the coercivity of Nd-Fe-B permanent magnets," vol. 22, no. 1, pp. 386–403. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/14686996.2021.1916377

[5] X. Shi, "Design optimization of insulation usage and space conditioning load using energy simulation and genetic algorithm," vol. 36, no. 3, pp. 1659–1667. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S036054421000770X

[6] L. Magnier and F. Haghighat, "Multiobjective optimization of building design using TRNSYS simulations, genetic algorithm, and Artificial Neural Network," vol. 45, no. 3, pp. 739–746. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360132309002091

[7] G. Lambard, T. Sasaki, K. Sodeyama, T. Ohkubo, and K. Hono, "Optimization of direct extrusion process for Nd-Fe-B magnets using active learning assisted by machine learning and Bayesian optimization," vol. 209, p. 114341. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1359646221006217

[8] Y. Wang, Y. Tian, T. Kirk, O. Laris, J. H. Ross, R. D. Noebe, V. Keylin, and R. Arróyave, "Accelerated design of Fe-based soft magnetic materials using machine learning and stochastic optimization," vol. 194, pp. 144–155. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1359645420303451

[9] J. J. Möller, W. Körner, G. Krugel, D. F. Urban, and C. Elsässer, "Compositional optimization of hard-magnetic phases with machine-learning models," vol. 153, pp.

53–61. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1359645418302490

[10] A. Eledeen, N. Neveu, M. Frey, Y. Huber, C. Mayes, and A. Adelmann, "Machine learning for orders of magnitude speedup in multiobjective optimization of particle accelerator systems," vol. 23, no. 4, p. 044601. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.23.044601

[11] H. Yamano, A. Kovacs, J. Fischbacher, K. Danno, Y. Umetani, T. Shoji, and T. Schrefl, "Efficient optimization approach for designing power device structure using machine learning," vol. 62, p. SC1050. [Online]. Available: https://iopscience.iop.org/article/10.35848/1347-4065/acb061

[12] B. Settles, "Active Learning Literature Survey." [Online]. Available: http://digital.library.wisc.edu/1793/60660

[13] T. M. Mitchell, *Machine Learning*, ser. McGraw-Hill Series in Computer Science. McGraw-Hill.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive Computation and Machine Learning. The MIT Press. [Online]. Available: https://doi.org/10.1007/s10710-017-9314-z

[15] N. Buduma and N. Locascio, *Fundamentals of Deep Learning*, 2nd ed. O'Reilly Media Inc.

[16] A. Kovacs, "Novel numerical studies of bit patterned magnetic recording." [Online]. Available: https://repositum.tuwien.at/handle/20.500.12708/16371?mode=full

[17] S. Boyd and L. Vandenberghe, *Convex Optimization*, 7th ed. Cambridge University Press. [Online]. Available: https://doi.org/10.1017/CBO9780511804441

[18] T. Santos and S. Xavier. A Convergence indicator for Multi-Objective Optimisation Algorithms. [Online]. Available: http://arxiv.org/abs/1810.12140

[19] H. Tamaki, H. Kita, and S. Kobayashi, "Multi-objective optimization by genetic algorithms: A review," in *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE, pp. 517–522. [Online]. Available: http://ieeexplore.ieee.org/document/542653/

[20] L. Caldas and L. Norford, "Genetic Algorithms for Optimization of Building Envelopes and the Design and Control of HVAC Systems," vol. 125, no. 3, pp. 343–351. [Online]. Available: https://asmedigitalcollection.asme.org/solarenergyengineering/article/125/3/343/463435/Genetic-Algorithms-for-Optimization-of-Building

[21] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, 15th ed. Elsevier, Academic Press.

[22] S. Ruder. An overview of gradient descent optimization algorithms. [Online]. Available: http://arxiv.org/abs/1609.04747

[23] L. Chen, K.-U. Bletzinger, N. R. Gauger, and Y. Ye. A gradient descent akin method for constrained optimization: Algorithms and applications. [Online]. Available: http://arxiv.org/abs/2302.11898

[24] C.-P. Cheng, C.-W. Liu, and C.-C. Liu, "Unit commitment by Lagrangian relaxation and genetic algorithms," vol. 15, no. 2, pp. 707–714. [Online]. Available: http://ieeexplore.ieee.org/document/867163/

[25] J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," vol. 8, pp. 89 497–89 509. [Online]. Available: https://ieeexplore.ieee.org/document/9078759/

[26] T. Li, G. Shao, W. Zuo, and S. Huang, "Genetic Algorithm for Building Optimization: State-of-the-Art Survey," in *Proceedings of the 9th International Conference on Machine Learning and Computing*. ACM, pp. 205–210. [Online]. Available: https://dl.acm.org/doi/10.1145/3055635.3056591

[27] L. Caldas, "Generation of energy-efficient architecture solutions applying GENE_ARCH: An evolution-based generative design system," vol. 22, no. 1, pp. 59–70. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1474034607000493

[28] R. Ooka and K. Komamura, "Optimal design method for building energy systems using genetic algorithms," vol. 44, no. 7, pp. 1538–1544. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360132308001820

[29] F. Pernodet, H. Lahmidi, and P. Michel, "USE OF GENETIC ALGORITHMS FOR MULTICRITERIA OPTIMIZATION OF BUILDING REFURBISHMENT."

[30] K. Deb, E. Goodman, C. A. Coello Coello, K. Klamroth, K. Miettinen, S. Mostaghim, and P. Reed, Eds., *Evolutionary Multi-Criterion Optimization: 10th International Conference, EMO 2019, East Lansing, MI, USA, March 10-13, 2019, Proceedings*, ser. Lecture Notes in Computer Science. Springer International Publishing, vol. 11411. [Online]. Available: http://link.springer.com/10.1007/978-3-030-12598-1

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Duchesnay, "Scikit-learn: Machine Learning in Python," vol. 12, no. 85, pp. 2825–2830. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner,

I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." [Online]. Available: https://www.tensorflow.org/

[33] P. Wittek, "Pattern Recognition and Neural Networks," in *Quantum Machine Learning*. Elsevier, pp. 63–71. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780128009536000062

[34] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," vol. 18, no. 7, pp. 1527–1554. [Online]. Available: https://direct.mit.edu/neco/article/18/7/1527-1554/7065

[35] V. Plevris, G. Solorzano, N. Bakas, and M. Ben Seghier, "Investigation of performance metrics in regression analysis and machine learning-based prediction models," in *8th European Congress on Computational Methods in Applied Sciences and Engineering*. CIMNE. [Online]. Available: https://www.scipedia.com/public/Plevris_et_al_2022a

[36] F. J. Anscombe, "Graphs in Statistical Analysis," vol. 27, no. 1, pp. 17–21. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00031305.1973.10478966

[37] D. Huang, T. Allen, W. Notz, and N. Zeng, "Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models," vol. 34, no. 3, pp. 441–466. [Online]. Available: https://doi.org/10.1007/s10898-005-2454-3

[38] A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola, "Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index," vol. 181, no. 2, pp. 259–270. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0010465509003087

[39] S. Burhenne, D. Jacob, and G. Henze, "Sampling based on Sobol' sequences for Monte Carlo techniques applied to building simulations," pp. 1816–1823. [Online]. Available: https://www.researchgate.net/publication/257139589_Sampling_based_on_Sobol%27_sequences_for_Monte_Carlo_techniques_applied_to_building_simulations

[40] J. Kuczynski, K. Tritz, M. Thoma, M. Newville, M. Kümmerer, M. Bolingbroke, M. Tartre, M. Pak, N. J. Smith, N. Nowaczyk, N. Shebanov, O. Pavlyk, P. A. Brodtkorb, P. Lee, R. T. McGibbon, R. Feldbauer, S. Lewis, S. Tygier, S. Sievert, S. Vigna, S. Peterson, S. More, T. Pudlik, T. Oshima, T. J. Pingel, T. P. Robitaille, T. Spura, T. R. Jones, T. Cera, T. Leslie, T. Zito, T. Krauss, U. Upadhyay, Y. O. Halchenko, and Y. Vázquez-Baeza, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," vol. 17, no. 3, pp. 261–272. [Online]. Available: https://www.nature.com/articles/s41592-019-0686-2

[41] I. M. Sobol', D. Asotsky, A. Kreinin, and S. Kucherenko, "Construction and Comparison of High-Dimensional Sobol' Generators," vol. 2011, no. 56, pp. 64–79. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/wilm.10056

[42] T. Sparks and S. Baird, "Self-driving-lab-demo," Sparks/Baird Materials Informatics. [Online]. Available: https://github.com/sparks-baird/self-driving-lab-demo

[43] M. Schoenauer and Z. Michalewicz, "Sphere operators and their applicability for constrained parameter optimization problems," in *Evolutionary Programming VII*. Springer Berlin Heidelberg, vol. 1447, pp. 239–250. [Online]. Available: http://link.springer.com/10.1007/BFb0040777

[44] A. Fozooni, O. Kamari, M. Pourtalebiyan, M. Gorgich, M. Khalilzadeh, and A. Valizadeh, "An Analysis of the Operation Factors of Three PSO-GA-ED Meta-Heuristic Search Methods for Solving a Single-Objective Optimization Problem," vol. 2022, pp. 1–22. [Online]. Available: https://www.hindawi.com/journals/cin/2022/2748215/

[45] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," vol. 6, no. 2, pp. 182–197. [Online]. Available: http://ieeexplore.ieee.org/document/996017/

[46] T. Odland and K. Subbarao, "Paretoset," tommyod. [Online]. Available: https://github.com/tommyod/paretoset

A. Appendix

A.1. Efficient Global Optimization (EGO) algorithm

This subsection explains how the EGO algorithm exploits its meta model in an iterative manner to solve an optimization problem with only a small number of expensive validations.

First, initial data points are costly obtained either from experimental measurements or from simulations. A meta model based on the dataset is fitted which is also called "response surface model". The objective and constraint functions of the optimization problem can be quite cheaply evaluated on the response surface model instead of the original model. [2, 37] These cheap function evaluations are only approximations of the actual function values, but they allow the algorithm to trade speed for accuracy. The algorithm determines which samples to validate next by following an "expected improvement function". Next, the original model is evaluated at the suggested points to validate the estimated optimum. If the solution is not satisfactory, the response surface model is refined by fitting more data in the proposed area to gain more accuracy. This refinement data can be obtained from the validation made in the step before. We iteratively continue this approach until the solution is good enough. [2]

A.2. Active Learning Scheme Configuration

The *engine* of the modular Python framework used in this thesis is configured with the following parameters. The first subset of the parameters deal with the setup of the parameter study. The second describes central configuration that was not changed over the course of the parameter study.

scheme	Optimization scheme for solving optimization problems. Either CO or ALS.
testProblem	Test problem used for optimization.
sampleMode	Sampling method.
n	Sample size for genetic algorithm's population and also training data for meta model in active learning scheme.
algorithm	The algorithm used for optimization. By default $NSGA-II$
max_n_iter	Maximum number of iterations for ALS. Used as upper boundary to stop the computation in any case. By default 100
max_n_gen	Maximum number of generations in the optimization algorithm. Used as upper boundary to stop the computation in any case. By default 100
max_n_eval	Maximum number of evaluations during optimization for algorithm. Computed by $max_neval = populationsizegenerations$
tuneInterval	Interval for tuning machine learning model hyperparameters. By default 1 to allow tuning in every iteration.
ftol	Termination criterion based on the change in the objective space. Translates to the distance between computed solution and optimum. By default 0.05.
learning_rate	This parameter determines the knowledge gain of the ALS in each iteration. It is the percentage of final algorithm population to be validated and added to the training data. By default $\lfloor 10\%$ of $n \rfloor$

Table A.1.: Scheme configuration set

B. Deutsche Zusammenfassung

Optimierungsprobleme sind allgegenwärtig in der Natur aber auch in der Wirtschaft, dem Ingenieurwesen usw. Sie stellen erhebliche Herausforderungen bei der Suche nach effizienten Lösungen dar, z.B. für Materialdesign oder zum Finden der kürzesten Verbindung. Die Komplexität dieser Probleme liegt oft daran, dass es mehrer Optimierungsziele gibt und dass die Zielfunktion einer verrauschten hochdimensionalen Landschaften gleicht. Die Optimierung dieser Probleme wird besonders schwierig, wenn die Bewertung von Zielfunktionen rechnerisch aufwendig ist, was dazu führt, dass die Berechnungsprozesse übermäßig viel Zeit und/oder Energie verbrauchen.

"Active learning schemes" (ALS) erwiesen sich als vielversprechender Ansatz, zur Lösung dieser Herausforderungen. Das ALS funktioniert, indem es iterativ ein Machine Learning Ersatz-Modell (eng. *surrogate model*) anlernt, um die Zielfunktion basierend auf neuen Datenpunkten, die es selbst ausgewählt und erkundet hat, zu approximieren. Dadurch wird der Optimierungsprozess mit jeder Iteration verbessert. Diese Arbeit untersucht die Wirksamkeit und Effizienz des ALS bei der Lösung komplexer Optimierungsaufgaben bei Einzel- und Mehrziel-Problemen.

Im Zuge dieser Arbeit wurde ein modulares Python-Framework entwickelt, um Experimente mit einem ALS in verschiedenen Konfigurationen durchzuführen. Der genetische Algorithmus NSGA-II dient als Maßstab, mit dem die Performance des ALS verglichen wird. Beide Lösungsansätze wurden in einer vergleichenden Parameterstudie untersucht. Diese Studie wurde mit verschiedenen Stichprobengrößen und vier verschiedenen Stichprobenverfahren bei sechs Optimierungstestfunktionen durchgeführt.

Die Forschung ergibt, dass das ALS in simplen Problemstellungen leicht etwa 4-10 Mal effektiver als der genetische Algorithmus ist. Jedoch führt seine begrenzte Fähigkeit, komplexe, verrauschte Funktionslandschaften zu erkunden, zu suboptimalen Lösungen. Die Ergebnisse zeigen, dass das ALS zwar viele Pareto-optimale Lösungen bei mehrzieligen Funktionen findet, es jedoch nur begrenzt fähig ist, das gesamte Pareto-Set gleichmäßig zu erkunden. Zusätzlich legen die Ergebnisse nahe, dass verschiedene Stichprobenverfahren nur eine untergeordnete Rolle in der Gesamteffizienz des Lösungsansatzes spielen (sowohl für den genetischen Algorithmus als auch für das ALS).

Daher präsentiert diese Arbeit mehrere Verbesserungsvorschläge, einschließlich der Integration von Unsicherheitsmaßen wie der Standardabweichung durch Gauß-Prozess Regressoren als Ersatzmodell und der Anwendung von Mini-Batching-Techniken im Modelltrainingsprozess. Weitere Verfeinerungen beinhalten beispielsweise einen Mechanismus zur Vermeidung der möglichen Neubewertung von Datenpunkten, welche in unmittelbarer Nähe von bereits bekannten Punkten liegen.

Diese Verbesserungen zielen darauf ab, die Erkundungsfähigkeit des ALS zu erweitern und seine Leistung in komplexen, ressourcenintensiven Optimierungsproblemen zu verbessern.

Zusammenfassend trägt diese Arbeit zum laufenden Diskurs in der Optimierungsforschung bei, indem sie die Fähigkeiten des ALS bei ein- und mehrzieliger Optimierung kritisch untersucht und potenzielle Wege für seine Verbesserung vorschlägt. Die präsentierten Ergebnisse dienen als Proof-of-Concept und bahnen den Weg für umfassendere und statistisch signifikante zukünftige Studien in diesem Bereich.