UNIVERSITY OF CALIFORNIA
Los Angeles

Studies in Hyperparameter Tuning, Design Selection and Optimization

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Statistics

by

Samuel O. Onyambu

2024

ABSTRACT OF THE DISSERTATION

Studies in Hyperparameter Tuning, Design Selection and Optimization

by

Samuel O. Onyambu
Doctor of Philosophy in Statistics
University of California, Los Angeles, 2024
Professor Hongquan Xu, Chair

This dissertation explores advanced optimization techniques, focusing on hyperparameter tuning, design strategy selection, and novel optimization methods. First, we investigate a modified Differential Evolution (DE) algorithm for generating uniform projection designs, emphasizing the importance of hyperparameter configuration. We analyze the surface structure of these hyperparameters and provide guidelines for optimizing settings under various conditions. Next, we examine the role of initial design choices in prediction and sequential optimization using Efficient Global Optimization (EGO), demonstrating that uniform projection designs outperform traditional strategies such as maximin distance designs, particularly in high-dimensional spaces. Finally, we introduce a Kriging-based sequential region shrinking method that integrates EGO to efficiently reduce the search space by targeting promising data points. Comparative results show that this method not only requires fewer computational resources than conventional tuning techniques like grid and random search, but also outperforms other Bayesian optimization methods such as TREGO. These findings offer significant contributions to the optimization field, enhancing both theoretical understanding and practical applications.

The dissertation of Samuel O. Onyambu is approved by

Mark S. Handcock

Jessica Jingyi Li

Qing Zhou

Hongquan Xu, Committee Chair

University of California, Los Angeles

2024

To my parents, Joseph and Jane, —

and my siblings, Tabitha, Naomi, Jasper, and Lilian,—

who among so many other things ensured that I gave my all to accomplish this task.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Professor Dr. Hongquan Xu, for his invaluable guidance and support throughout my PhD journey. His insights and encouragement have been essential in shaping this dissertation.

I am also thankful to my committee members for their helpful feedback and advice. I extend my thanks to my colleagues for their collaboration and to my friends and family for their unwavering support during this time.

Special thanks to the Civil and Environmental Engineering Department at UCLA for providing financial support through the Graduate Student Researcher (GSR) position, under the supervision of Professor Dr. Tierra Bills.

VITA

2011–2015 B.S. (Statistics and Programming), Kenyatta University, Nairobi, Kenya.

2016–2018 M.S. (Statistics), California State University, Fullerton (CSUF).

2016–2018 Teaching Assistant, Mathematics Department, CSUF.

2018–2019 Dorothy Radcliffe Dee Fellowship

2018-2020 Irma Polaski Fellowship

2019–2021 Teaching Assistant, Statistics Department, UCLA.

2021 Summer Mentored Research Fellowship

2021–2022 Teaching Associate, Statistics Department, UCLA.

2022 Summer Graduate Student Researcher, Department of Statistics, UCLA.

2022–2023 Teaching Fellow, Statistics Department, UCLA.

2023–2024 Graduate Student Researcher, Civil and Environmental Engineering Department, UCLA.

2024 Summer Graduate Student Researcher, Department of Statistics, UCLA.

PUBLICATIONS

Laxman Dahal, Henry Burton, and Samuel Onyambu (2022). "Quantifying the effect of probability model misspecification in seismic collapse risk assessment". In: *Structural Safety* 96, p. 102185

CHAPTER 1

Introduction

1.1 Introduction to Optimization Algorithms

Optimization algorithms play a pivotal role in solving complex problems where analytical solutions are either unavailable or computationally infeasible. Among these, gradient-free algorithms are particularly valuable for functions that are non-differentiable, discontinuous, or noisy. Two prominent approaches in this category are metaheuristic algorithms and Bayesian optimization, each offering unique strengths and applications.

Metaheuristic algorithms are inspired by natural processes such as evolution, swarm behavior, or physical phenomena. Popular examples include Differential Evolution, Genetic Algorithms, Particle Swarm Optimization and Simulated Annealing. These algorithms excel in exploring vast and complex solution spaces, making them highly effective for global optimization. Their stochastic nature and population-based search strategies allow them to escape local optima and adapt to diverse problem landscapes. On the other hand, Bayesian optimization takes a probabilistic approach, using surrogate models (such as Gaussian processes) to approximate the objective function. By sequentially selecting points that balance exploration and exploitation, Bayesian optimization achieves efficiency in scenarios where function evaluations are expensive.

Both approaches are integral to my research, offering robust, gradient-free methods to address the challenges of experimental design construction and data-driven model optimization.

1.2 Metaheuristic Algorithms

Metaheuristic algorithms are high-level problem-solving frameworks designed to find near-optimal solutions to complex optimization problems. These algorithms are particularly useful for problems where traditional optimization methods may struggle due to factors like high-dimensional search spaces, non-linearity, discontinuity, or multimodality. The key characteristics of metaheuristic algorithms include:

1. **General-Purpose**: Applicable to a wide range of problems without requiring problem-specific knowledge.
2. **Approximate Solutions**: Focus on finding good-enough solutions in a reasonable time rather than guaranteed optimal solutions.
3. **Stochastic Nature**: Often incorporate randomization to explore the search space and avoid local optima.
4. **Iterative Process**: Evolve candidate solutions over several iterations using a combination of exploration (searching broadly) and exploitation (refining promising areas).

1.2.1 Evolutionary Algorithms

Evolutionary algorithms are a family of optimization techniques inspired by the principles of natural evolution, such as selection, mutation, recombination or crossover, and survival of the fittest. These algorithms operate on a population of candidate solutions, iteratively improving them based on a defined fitness function. Common examples include Genetic Algorithms (GA), Differential Evolution (DE), and Evolution Strategies (ES), each with unique mechanisms for generating and selecting solutions.

1.2.1.1 Genetic Algorithm (GA)

Genetic Algorithms (GAs) are search heuristics inspired by the principles of natural selection and genetics, first introduced by Holland (1975). In GAs, a population of candidate solutions, called chromosomes, evolves over iterations, or generations, through processes like selection, crossover, and mutation. The algorithm aims to find an optimal or near-optimal solution to a given problem by mimicking the survival-of-the-fittest principle in biological systems (Mitchell 1998).

The GA process begins with the initialization of a population, typically generated randomly within the solution space. Each chromosome in the population is evaluated using a fitness function, which quantifies the quality of the solution it represents. Based on fitness scores, a selection mechanism such as roulette wheel or tournament selection is used to choose parent chromosomes for reproduction. The better the fitness, the more likely a chromosome is selected, promoting better solutions (Kawachi and Ando 1992).

Reproduction involves crossover, where segments of parent chromosomes are combined to produce offspring. Common crossover methods include single-point, two-point, and uniform crossover. This recombination introduces new solution candidates into the population. Additionally, mutation is applied to offspring with a small probability to alter random genes, maintaining diversity and preventing premature convergence to local optima (Jong Kenneth 1975).

The GA process iterates through selection, crossover, mutation, and fitness evaluation until a termination criterion is met, such as a maximum number of generations or convergence to a solution. GAs have been successfully applied to various optimization problems, including scheduling, engineering design, and machine learning hyperparameter tuning. Their adaptability and ability to handle complex, non-linear problems have made them a popular choice in optimization research (Whitley 1994).

However, GAs face challenges like computational cost due to their population-based nature and sensitivity to parameter settings, such as mutation rate, crossover probability, and

population size. Hybrid approaches combining GAs with other optimization methods, like local search or simulated annealing, have been developed to address these issues, enhancing convergence and efficiency (Michalewicz 2013).

1.2.1.2 Differential Evolution (DE)

Differential Evolution (DE) is a population-based optimization algorithm introduced by Storn and Price (1997), specifically designed for continuous optimization problems. Unlike GAs, DE focuses on vector differences to guide its search, making it particularly effective for high-dimensional and non-linear optimization tasks. DE is simple yet powerful, with few parameters to tune (Price, Storn, and Lampinen 2006).

The DE process starts with initializing a population of candidate solutions as vectors within the solution space. Each vector is evaluated using an objective function to determine its fitness. The algorithm then generates trial vectors for each population member through mutation, crossover, and selection. Mutation in DE involves creating a donor vector by adding the scaled difference between two randomly chosen population vectors to a third vector, a strategy unique to DE (Mezura-Montes, Velázquez-Reyes, and Coello Coello 2006).

Crossover combines the donor vector with the target vector to produce a trial vector. This process can be uniform or binomial, depending on whether crossover is controlled by a fixed or probabilistic scheme. The trial vector is evaluated, and if it outperforms the target vector in terms of fitness, it replaces the target in the population for the next generation. This selection mechanism ensures steady improvement in the population's quality over iterations (Das and Suganthan 2010).

DE's strength lies in its robustness and ability to balance exploration and exploitation through its mutation and selection strategies. Its simple structure and limited parameter requirements, primarily the mutation factor and crossover rate, make it easy to implement and tune. DE has been applied to diverse areas, including function optimization, neural network training, and control system design (Miettinen 1999).

Despite its effectiveness, DE may face challenges in multimodal optimization problems, where it can become trapped in local optima. Variants of DE, such as adaptive DE and self-adaptive DE, have been developed to address these issues by dynamically adjusting parameters during the search process. Hybrid approaches integrating DE with other techniques, like gradient-based methods, have also shown promise in improving convergence and tackling complex optimization problems (Qin, Huang, and Suganthan 2008).

1.2.1.3 Evolution Strategies (ES)

Evolution Strategies (ES) are a prominent subclass of evolutionary algorithms tailored for optimizing real-valued continuous functions. Developed in the 1960s by Ingo Rechenberg and Hans-Paul Schwefel, these algorithms are inspired by natural evolution, focusing on adaptation and mutation while eschewing crossover mechanisms typical in other evolutionary approaches like Genetic Algorithms (Schwefel 1977; Vent 1975). ES are population-based, relying on iterative cycles of mutation, selection, and reproduction to refine candidate solutions. Their robustness, simplicity, and adaptability make them particularly effective for solving high-dimensional, noisy, or non-linear optimization problems (Beyer and Schwefel 2002).

The standard ES workflow begins with a population of candidate solutions, each represented as a vector of real-valued parameters. Mutation, the core variation operator, introduces randomness by perturbing these parameters using Gaussian noise. Selection mechanisms, such as $(\mu+\lambda)$ or (μ, λ), then choose the fittest individuals to form the next generation. While $(\mu + \lambda)$ strategies retain both parents and offspring for survival, promoting stability, (μ, λ) strategies consider only offspring, encouraging greater exploration of the search space (Schwefel 1993). These mechanisms make ES versatile in balancing exploration and exploitation.

A significant extension of ES is the Directed Variation, introduced by Zhou and Li (2003), which enhances the mutation process by guiding it with directional information derived from

the problem landscape (Zhou and Li 2003). Unlike standard random isotropic mutations, directed variation biases mutation steps toward promising regions of the search space. Zhou and Li proposed frameworks for incorporating directional probabilities or fitness gradients into the mutation process, improving convergence rates and solution quality in complex optimization problems. This refinement allows ES to adapt dynamically to the fitness landscape, avoiding premature convergence while maintaining effective exploration.

The self-adaptation mechanism in ES, developed in earlier works (Schwefel 1981), complements directed variation by evolving strategy parameters, such as mutation step sizes, alongside candidate solutions. This approach enables the algorithm to adjust its search behavior based on the landscape's characteristics. When combined with directed variation, self-adaptation can achieve even better convergence properties by synergizing global exploration with locally informed refinement (Beyer and Arnold 2001). Directed variation thus enriches the evolutionary process by incorporating both dynamic parameter adjustment and problem-specific guidance.

Directed variation is particularly advantageous in applications involving constrained or multi-modal optimization. Zhou and Li's work highlights its efficacy in improving optimization outcomes for tasks with complex, irregular landscapes (Zhou and Li 2003). By biasing mutations toward beneficial directions, ES with directed variation can escape local optima and converge efficiently to global optima. This capability has been demonstrated in fields such as engineering design, robotics, and neural network training.

The theoretical foundation of directed variation draws on insights from gradient-based optimization while retaining the flexibility of stochastic methods. Zhou and Li's approach blends deterministic directional cues with stochastic exploration, bridging the gap between classical optimization techniques and evolutionary computation. This hybrid methodology underscores the broader adaptability of ES, allowing it to tackle a diverse range of optimization challenges (Beyer and Schwefel 2002; Zhou and Li 2003).

Another notable advancement is Covariance Matrix Adaptation (CMA-ES), introduced

by Hansen and Ostermeier (2001). CMA-ES dynamically adjusts the covariance matrix of the mutation distribution to align with the topology of the fitness function. This technique enables ES to search more effectively along relevant dimensions of the landscape, particularly in ill-conditioned or anisotropic problems.

In summary, ES, enriched with innovations like directed variation, self-adaptation, and CMA-ES, offer a robust and versatile framework for optimization. Together, these advancements have solidified ES as a cornerstone of evolutionary computation, enabling it to excel in solving real-world problems with complex, high-dimensional, and noisy landscapes.

1.2.2 Swarm intelligence (SI)

Swarm intelligence algorithms are optimization techniques inspired by the collective behavior of decentralized, self-organized systems in nature, such as flocks of birds, schools of fish, or ant colonies. These algorithms use the interaction of simple agents to solve complex problems without central control. Swarm intelligence models excel in parallelism, adaptability, and robustness, making them well-suited for solving optimization problems across diverse domains, including engineering, scheduling, and machine learning (Kennedy and Eberhart 1995).

1.2.2.1 Particle Swarm Optimization (PSO)

Introduced by Kennedy and Eberhart (1995), is inspired by the flocking behavior of birds and schooling of fish. In PSO, each individual in the swarm, called a particle, represents a candidate solution. The particles explore the solution space by updating their positions based on their own best-known position, the best-known position of their neighbors, and the overall global best position. These updates are guided by two key parameters: cognitive and social components, which balance personal learning and group influence.

The mathematical foundation of PSO lies in velocity and position updates. A particle's velocity is influenced by three components: inertia, which maintains momentum; the cogni-

tive term, which pulls the particle toward its own best position; and the social term, which attracts the particle toward the global best position. By iteratively updating positions and velocities, PSO achieves convergence to near-optimal solutions. Its simplicity and efficiency have made it popular for a wide range of problems, from neural network training to resource allocation (Clerc and Kennedy 2002).

One of the primary advantages of PSO is its ease of implementation, as it requires fewer hyperparameters compared to other optimization algorithms. However, it faces challenges such as premature convergence, especially in high-dimensional or multimodal optimization problems. Variants of PSO, such as constriction coefficient PSO and adaptive PSO, have been developed to address these limitations and improve exploration and exploitation (Shi and Eberhart 1998).

1.2.2.2 Ant Colony Optimization (ACO)

Another prominent SI algorithm, was introduced by Dorigo (1996) and is inspired by the foraging behavior of ants. Ants use pheromone trails to communicate and collectively find optimal paths between their nest and a food source. In ACO, artificial ants construct solutions by traversing a graph representation of the problem, depositing pheromones on paths to indicate their quality.

The algorithm operates iteratively, where ants build solutions based on probabilistic rules influenced by pheromone levels and heuristic information, such as edge lengths or costs. After constructing solutions, pheromones are updated: stronger paths receive more pheromone reinforcement, while weaker paths evaporate over time. This dynamic update mechanism balances exploration of new paths and exploitation of promising ones, allowing ACO to identify optimal or near-optimal solutions (Dorigo 2007).

ACO has been successfully applied to combinatorial optimization problems, such as the Traveling Salesman Problem, vehicle routing, and scheduling. Its strengths lie in its adaptability and ability to integrate heuristic knowledge. However, ACO's performance depends

on parameters like evaporation rate, pheromone influence, and heuristic weighting, which require careful tuning to avoid premature convergence or excessive exploration (Blum 2005).

Despite their differences, both PSO and ACO share a reliance on decentralized control and emergent behavior to solve problems. While PSO excels in continuous optimization due to its particle dynamics, ACO is better suited for discrete optimization tasks because of its graph-based representation. Hybrid approaches combining PSO and ACO have been developed to leverage the strengths of both algorithms, particularly for problems with mixed discrete and continuous variables (Talbi 2009).

Swarm intelligence algorithms, including PSO and ACO, are inherently parallelizable, making them attractive for large-scale optimization problems. Their ability to balance exploration and exploitation is a key factor in their success. Recent research has focused on adapting these algorithms to dynamic and multi-objective optimization problems, where solution landscapes change over time or involve conflicting objectives (Coello, Pulido, and Lechuga 2004).

One challenge in swarm intelligence algorithms is their computational complexity, especially for high-dimensional problems or large populations. To address this, researchers have developed variants like distributed swarm intelligence and hybrid metaheuristics. These approaches aim to reduce computational cost while maintaining solution quality, enabling the application of swarm intelligence in fields like robotics, logistics, and bioinformatics.

Moreover, advances in hardware, such as GPUs and parallel computing platforms, have further enhanced the scalability of swarm intelligence algorithms. These developments allow for real-time applications in domains such as autonomous navigation, sensor networks, and real-time scheduling, where traditional optimization methods may fall short (Yang and Karamanoglu 2013).

In conclusion, swarm intelligence algorithms like PSO and ACO have revolutionized optimization by drawing inspiration from nature. Their versatility, adaptability, and robustness have enabled solutions to a wide range of complex problems. With ongoing advancements

in hybridization, parameter tuning, and computational power, swarm intelligence continues to be a cornerstone of modern optimization research.

1.2.3 Simulated Annealing (SA)

Is a probabilistic optimization technique inspired by the annealing process in metallurgy, where materials are heated and slowly cooled to alter their physical properties. Proposed by Kirkpatrick, Gelatt Jr, and Vecchi (1983), SA solves complex optimization problems by mimicking this physical process, exploring a solution space to find near-optimal or global optima. Unlike traditional optimization methods, it allows temporary acceptance of worse solutions to escape local minima, making it especially effective for problems with rugged landscapes.

The algorithm operates by iteratively modifying a candidate solution and evaluating its quality using a predefined objective function. A key feature of SA is its acceptance criterion, which is governed by a probabilistic function dependent on temperature and the difference in solution quality. This function, often based on the Metropolis criterion, reduces the likelihood of accepting worse solutions as the temperature decreases. This simulated "cooling" process, if managed properly, ensures convergence to an optimal solution while maintaining exploration of the solution space early on (Aarts and Korst 1989).

Temperature scheduling plays a critical role in SA's effectiveness. Commonly, exponential decay is used, where the temperature decreases geometrically over iterations. However, alternative cooling schedules like linear or logarithmic decay can also be employed, each with unique trade-offs in convergence speed and solution quality. The choice of cooling schedule and initial temperature significantly affects performance, as improper tuning may lead to suboptimal solutions or slow convergence.

SA has been successfully applied to a wide range of problems, including combinatorial optimization tasks like the Traveling Salesman Problem (TSP), job scheduling, and graph partitioning. For instance, Johnson et al. (1991) demonstrated SA's efficiency in solving the

TSP by leveraging its capability to explore and exploit the problem's complex search space. In continuous optimization, modifications to the algorithm have enabled its use in problems like parameter estimation and engineering design.

While SA has many strengths, it also faces challenges, such as the need for careful parameter tuning and the potential for slow convergence. Hybrid approaches combining SA with other optimization techniques, such as genetic algorithms or gradient-based methods, have been proposed to address these issues. Such hybrid algorithms often leverage SA's exploration capabilities while benefiting from the exploitation strengths of complementary methods (Yang 2010).

Theoretical work on SA has shown that it can asymptotically converge to the global optimum under certain conditions. Specifically, if the cooling schedule is sufficiently slow, the algorithm can theoretically explore all possible solutions and guarantee convergence. However, these conditions are often impractical due to computational constraints, necessitating heuristic adjustments in real-world applications (Hajek 1988).

In conclusion, simulated annealing remains a versatile and robust optimization technique suitable for a wide array of applications. Its success lies in its balance between exploration and exploitation, as well as its capacity to escape local minima. Advances in hybrid approaches and theoretical insights continue to expand its scope and effectiveness, making it a critical tool in modern optimization practices.

1.3 Bayesian Optimization

Beyond the metaheuristic optimization methods lies the notion of Bayesian Optimization. Bayesian optimization has become a widely adopted approach for black-box optimization problems, particularly for its sequential nature in handling noisy and non-convex functions commonly encountered in real-world scenarios. The framework often relies on Efficient Global Optimization (EGO), a sequential strategy that iteratively searches for optimal solu-

tions by balancing exploration of the search space and exploitation of known high-performing regions (Jones, Schonlau, and Welch 1998). At each step, Bayesian optimization evaluates new points based on their potential to improve the objective function while accounting for the uncertainty of the model predictions. This iterative, or sequential, nature allows Bayesian optimization to progressively refine its estimates, focusing on promising regions and thereby enhancing efficiency.

In Bayesian optimization, a Gaussian Process (GP) is typically used as the surrogate model, offering a flexible and powerful approach for modeling complex functions (Rasmussen and Williams 2006). GPs are particularly valuable because they predict the objective function's value at unexplored points and provide an estimate of prediction uncertainty, which can guide decisions on where to sample next. The choice of acquisition function, also known as the utility function, is central to Bayesian optimization, as it defines the criteria for selecting the next sample point. Common acquisition functions include Probability of Improvement (PI), Expected Improvement (EI), Upper Confidence Bound (UCB), and Lower Confidence Bound (LCB), each designed to guide the search process differently depending on the optimization strategy.

The Probability of Improvement (PI) function, for instance, favors points likely to outperform the current best-known solution, focusing on regions with high potential for incremental gains. Expected Improvement (EI), on the other hand, seeks points that offer the highest potential for improvement by integrating both the predicted mean and the uncertainty of the GP model, balancing exploration and exploitation more effectively (Mockus 1994). The Upper Confidence Bound (UCB) and Lower Confidence Bound (LCB) functions apply a degree of confidence to the GP predictions, where UCB emphasizes exploration by targeting areas with high uncertainty, while LCB may be used to address risk-averse or cost-sensitive problems by minimizing potential loss (Srinivas et al. 2009).

These acquisition functions collectively enable Bayesian optimization to adapt dynamically based on the task requirements. Each function aligns with different optimization

objectives, allowing Bayesian optimization to tailor its approach to complex landscapes and focus computational resources on the most promising areas of the search space. This adaptability, coupled with the sequential nature of EGO, makes Bayesian optimization a powerful method for tackling diverse and challenging optimization problems.

1.4 Leveraging Differential Evolution and Bayesian Optimization

As previously discussed, Differential Evolution (DE) is a versatile metaheuristic algorithm primarily used for optimizing continuous functions. Its simplicity and efficiency make it appealing across a wide range of applications, including experimental design. Notably, its strong global search capabilities, robust performance, and flexibility across diverse problem domains (Storn and Price 1997), combined with its high convergence speed for specific challenges (Babu and Jehan 2003) and inherent parallelizability (Kukkonen and Lampinen 2006), have established it as a highly favorable choice for design construction.

However, using Differential Evolution for discrete data tasks, such as experimental design construction, presents challenges. Applying DE in such cases requires modifications to its foundational structure to effectively manage discrete variables and discrete search space. Recent work by Stokes, Wong, and Xu (2024)introduced a modified DE algorithm specifically tailored for these discrete tasks, highlighting the need for adaptations to extend DE's functionality beyond continuous domains. Despite these advancements, the algorithm's performance remains highly sensitive to several hyperparameters, which play a critical role in influencing outcomes.

This sensitivity to hyperparameters emphasizes the importance of understanding and exploring the hyperparameter landscape. Developing optimized settings across different problem setups is essential for achieving reliable and efficient performance. Tailoring hyperparameters for specific tasks or datasets can greatly improve outcomes, but doing so requires a structured approach to hyperparameter tuning. As such, there is a need for strategies that systematically optimize hyperparameter configurations to align with varied and complex

problem requirements.

In this dissertation, we employ the DE algorithm and a novel localized region shrinkage Bayesian optimization in the construction of the uniform projection designs. Three studies are done and the structure of these studies is laid as follows.

The first study delves into the construction of space-filling designs focusing mainly on Uniform Projection Designs (UPDs) by investigating the surface structure of DE's hyper-parameters and their respective contributions. UPDs, introduced by Sun, Wang, and Xu (2019), are a special class of space-filling designs that ensure an even distribution of sample points across all lower-dimensional projections of a high-dimensional design space. By analyzing the impact of hyperparameters on the performance of the modified DE algorithm, the study aims to derive optimal settings for generating efficient UPDs using a second-order model. Through comparisons of various experimental designs and surrogate models, the research provides crucial guidelines for enhancing DE's performance. The insights gained from this study are intended to equip practitioners with effective strategies for optimizing hyperparameter configurations in practical applications.

Another critical aspect of optimization algorithms is the initial design choices, which can significantly impact both prediction accuracy and the efficiency of sequential optimization processes. The second study utilizes gaussian process and various initial experimental designs to model test functions determining the prediction power of the initial design. In addition, the effect of the initial design on the optimization performance via active learning is evaluated. This study highlights how initial design strategies greatly influence prediction prowess and early optimization results, with their effects diminishing as iterations proceed toward the global optimum. In particular, the research demonstrates that distance-based designs, like maximin distance designs, struggle in high-dimensional settings, whereas Uniform Projection Designs consistently perform well across varying dimensionalities.

In response to the challenges associated with traditional optimization approaches, the third study introduces a novel Kriging-based sequential region shrinking method, which ef-

fectively incorporates the EGO algorithm. This innovative method aims to progressively reduce the region of interest by focusing on the most promising data points during each iteration. The efficiency of this approach is validated through its application to various well-known physical test functions, where it significantly reduces the computational resources required compared to traditional hyperparameter tuning techniques like grid and random search. Furthermore, it demonstrates advantages over other Bayesian optimization strategies, such as TREGO, highlighting its practicality in resource-constrained environments.

Together, these studies provide a comprehensive exploration of advanced optimization techniques, ranging from hyperparameter tuning in DE to effective design strategy selection, culminating in novel optimization methods. The findings contribute valuable insights to both theoretical research and practical applications, showcasing the evolving landscape of optimization strategies. By bridging the gaps in existing methodologies, this work aims to enhance the efficacy of optimization processes across diverse fields, ultimately leading to more robust and efficient solutions for complex challenges.

CHAPTER 2

Tuning Differential Evolution Algorithm for Constructing Uniform Projection Designs

2.1 Introduction

Differential Evolution (DE) is a nature-inspired optimization algorithm that mimics the process of natural selection to efficiently search for optimal solutions within large search spaces. Since its introduction in 1997, DE has gained widespread popularity and has been extensively utilized to solve complex optimization problems. Its versatility and straightforward implementation have facilitated its integration into a multitude of real-world applications, ranging from parameter tuning in machine learning algorithms to engineering design optimization and financial portfolio management. DE has consistently demonstrated its efficacy across diverse problem domains, making it a preferred choice among researchers and practitioners alike due to its adaptability to various problem structures.

A particularly compelling application of the DE algorithm is in the realm of design generation. Recently, Stokes, Wong, and Xu (2024) proposed a DE-based approach for constructing order-of-addition designs, showcasing its efficiency compared to other metaheuristic algorithms, such as Simulated Annealing, Threshold Accepting, Genetic Algorithms, and Particle Swarm Optimization. Inspired by their findings, we adapt and extend their DE algorithm for the construction of Uniform Projection Designs (UPDs). UPDs, introduced by Sun, Wang, and Xu (2019), are specialized space-filling designs characterized by robust performance across various design criteria and impressive space-filling properties in multi-

ple dimensions. However, existing algorithms for generating UPDs remain underexplored, highlighting an important area for further research.

The performance of the DE algorithm is significantly influenced by its hyperparameters, which dictate the learning process of the optimization (Price, Storn, and Lampinen 2006). An inappropriate setting of these hyperparameters can lead to suboptimal performance of the DE algorithm. While the DE algorithm proposed by Stokes, Wong, and Xu (2024) encompasses several hyperparameters, their effects remain largely unexamined. Therefore, we aim to conduct a comprehensive study of the hyperparameters to elucidate their impacts on the algorithm's performance, providing insights that could enhance its effectiveness.

The challenge of determining the optimal hyperparameter settings for any learning process has been widely studied. Two primary frameworks dominate this landscape: the model-based framework and the model-free framework. Model-based hyperparameter optimization focuses on tuning hyperparameters by approximating the true learning algorithm, while model-free methods approach the optimization problem without parametric assumptions. Relevant literature on model-based hyperparameter optimization includes works by Falkner, Klein, and Hutter (2018), Hutter, Hoos, and Leyton-Brown (2011), Li et al. (2018), Lujan-Moreno et al. (2018), Mockus, Tiesis, and Zilinskas (1978), Snoek, Larochelle, and Adams (2012), Wu, Chen, and Liu (2020), and Zoph and Le (2016). In contrast, model-free frameworks encompass techniques such as manual search, grid search, random search, genetic algorithms, and orthogonal array tuning methods (Liashchynskyi and Liashchynskyi 2019).

Our approach leverages various types of designs and models to investigate the effectiveness of DE in constructing UPDs. Specifically, we aim to address three objectives: (i) identifying useful design types, (ii) determining the most effective models, and (iii) developing an efficient algorithm for the construction of UPDs. We employ different types of models to evaluate the performance of different designs, enabling us to visualize the response surface of the DE algorithm's hyperparameters. This framework outlines the data generation, modeling, and analysis procedures. The insights gained from our analysis will provide a gen-

eral guideline for optimal hyperparameter settings necessary for generating superior uniform projection designs.

The primary goal of this paper is to tune the DE algorithm for constructing UPDs through a detailed analysis of the response surface of the DE hyperparameters. This analysis involves studying how variations in hyperparameters influence the efficiency and optimality of the generated designs. By utilizing various designs to establish hyperparameter settings for DE optimization, we can identify critical factors or combinations of factors that contribute to enhanced performance. Our objective is to thoroughly explore the hyperparameter space of the DE algorithm using established designs to optimize the generation of UPDs. This approach is quite different from the Lujan-Moreno et al. (2018) method which used a 2^k factorial design with the response surface method (RSM) and ridge regression for screening to select the important factors in the data. While Lujan-Moreno et al. (2018) focuses on factor screening and selection with the traditional RSM approach, we emphasize on the comparisons of different types of designs and surrogate models in approximating the surface structure of the DE algorithm.

2.2 Differential Evolution Algorithm

Originating from the pioneering work of Storn and Price (1997), Differential Evolution (DE) has emerged as a powerful heuristic optimization algorithm, drawing inspiration from the mechanisms of biological evolution. To simulate survival-of-the-fittest dynamics, DE treats each candidate or agent as a chromosome made up of several genes and implements mutation and crossover procedures that allow beneficial genes to persist into future generations (Storn and Price 1997). DE operates on the principle of population-based search, where a set of candidate solutions evolves over successive generations towards optimal or near-optimal solutions. At its core, DE employs mutation, crossover, and selection operators to iteratively improve the quality of solutions. The algorithm's efficacy stems from its robustness, simplicity, and ability to handle non-linear, non-convex, and noisy optimization landscapes.

Without loss of generality, we assume that we want to minimize a real-valued objective function ϕ over an m-dimension space Ω. It has five steps

1. **Genetic Representation**: Let $\pi_1, \ldots, \pi_N$ be the initial population, where each agent $\pi_i = (\pi_{i1}, \ldots, \pi_{im})$ is randomly chosen from Ω.

2. **Mutation**: Mutation expands the search space of the current population. For each $i = 1, \ldots, N$, mutation produces a potential donor ν_i in Ω by adding the weighted difference of two agents to a third, all randomly chosen and distinct from the target (π_i), that is,

$$\nu_i = \pi_a + w(\pi_b - \pi_c) \tag{2.1}$$

   where $a \neq b \neq c$ are randomly chosen three distinct numbers from $1, \ldots, N$, and they are all different from i.

3. **Crossover**: Crossover blends the current generation of agents with the population of potential donors in order to form candidates for the next generation known as trial agents. For each $i = 1, \ldots, N$, one of the m variables of ν_i is randomly selected to directly enter the trial agent u_i. In this way, one variable is forced to change so that each u_i will certainly differ from its original target π_i. Next, with probability pCR, more variables are taken from ν_i and placed in the trial agent. Whichever variables do not take their value from the donor inherit their original value from π_i. Assuming j_0 is a random number from $1, \ldots, m$, this process can be written as follows: for $j = 1, ..., m$,

$$u_{ij} = \begin{cases} \nu_{ij} & \text{with probability pCR or if } j = j_0, \\ \pi_{ij} & \text{otherwise} \end{cases}$$

4. **Selection**: Selection creates the next generation of agents by comparing each target to its respective trial agent. The trial agent is adopted if it leads to an improvement

and is discarded otherwise. For minimization problems, this process is given by,

$$\pi_i = \begin{cases} u_i & \text{if } \phi(u_i) < \phi(\pi_i) \\ \pi_i & \text{otherwise} \end{cases}$$

5. **Repeat**: Repeat steps 2 through 4 over many generations until a specified stopping condition is satisfied.

Though quite simplistic, in order to use the algorithm for design generation, a modification is needed. This is because the designs lie on a discrete and constrained space. We must therefore ensure that the resulting mutated design is feasible. Thus a viable mutation function which ensures that the generated design is in the feasible region is needed. For example, suppose we have $n \times m$ Latin hypercube design (LHD), we need a mutation function such that once we mutate, we still end up with an $n \times m$ LHD. This calls for a modification of the mutation step. We use the modification as presented by Stokes, Wong, and Xu (2024). They proposed a perturbation modification.

Interpreting the second part of equation (2.1), $w(\pi_b - \pi_c)$, as adding some perturbations to π_a, Stokes, Wong, and Xu (2024) rephrased the mutation operation as follows:

$$\nu_i = \text{Perturb}(\pi_a), \tag{2.2}$$

whereby the perturbation is done by randomly swapping some elements in π_{aj} independently for each $j = 1, \ldots, m$. Letting pMut to be a small constant from [0, 1], they defined the perturbation operation as:

Perturb(π_a) : randomly swap 2 elements in π_{aj} with probability pMut for $j = 1, \ldots, m$.

With different choices of a they ended up with different variants for DE. In all the variants the choices of the global best, current agent and a random agent are fixed at a particular probability.

The modified version of the DE algorithm that Stokes, Wong, and Xu (2024) proposed contained six hyperparameters of interest:

- NP - The size of the population (N).
- itermax - the maximum number of iterations used.
- pCR - Probability of crossover.
- pMut - Probability of mutation.
- pGBest - Probability of using the global best for mutation.
- pSelf - Probability of using the current agent for mutation.

The first two hyperparameters, NP and itermax, determine the computational complexity, whereas the other four hyperparameters affect the evolution process. The two hyperparameters, pGBest and pSelf, determine the probability of using the global best and the current agent in the mutation (2.2), respectively. There is a constraint between these two hyperparameters, that is, pGBest + pSelf ≤ 1. The question that arises is how these hyperparameters interact with each other. Also whether we can do better than the proposed fixed probabilities to obtain the better settings for the DE algorithm for design generation. In this study, we consider values between $[10, 100]$ for N, $[500, 1500]$ for itermax, and $[0.05, 0.95]$ for pCR, pMut and pGBest. We fix pSelf = (1 - pGBest)/2 so that there is an equal chance for selecting a current agent and a random agent if the global best is not used.

2.3 Designs for Hyperparameter Settings

Various designs can be used to set the DE hyperparameters before the optimization process. As the functions optimized by DE are often complex with many local minima, one has to carefully choose the initial point for the optimization process. These initial points are determined using any of the methods discussed below. In each subsection below one method is described, and its benefits and drawbacks are discussed.

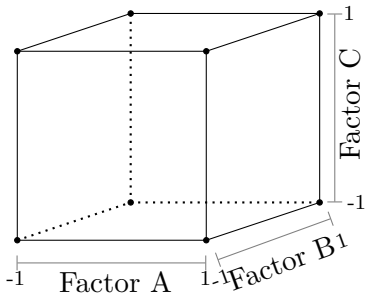


Figure 2.1: Geometric illustration of a 2^3 full factorial design

Full factorial designs (FFD)

Factorial designs are a research method for studying the effects of multiple independent variables on a response variable, formalized by Sir Ronald A. Fisher in the early 20th century (Fisher 1935). These designs typically involve defining factors at two or three levels, forming a grid of all possible combinations, resulting in 2^m or 3^m observations for m factors. Figure 2.1 shows a 2^3 full factorial design.

Full factorial designs sample points at the corners of a hypercube, ensuring uniform distribution across the design space. They allow for the analysis of main effects and interactions (Montgomery 2017), but can be complex and require larger sample sizes for adequate power (Tabachnick and Fidell 2019).

To mitigate the need for larger samples, fractional factorial designs were introduced by David John Finney (Finney 1945). These designs use a fraction of runs based on the sparsity-of-effects principle, focusing on main effects and lower-order interactions. They are expressed as 2^{m-p} or 3^{m-p}, depending on the factors set as products of others. Selecting defining relations for fractional designs is essential, with criteria such as maximum resolution and minimum aberration guiding this process (Wu and Hamada 2011).

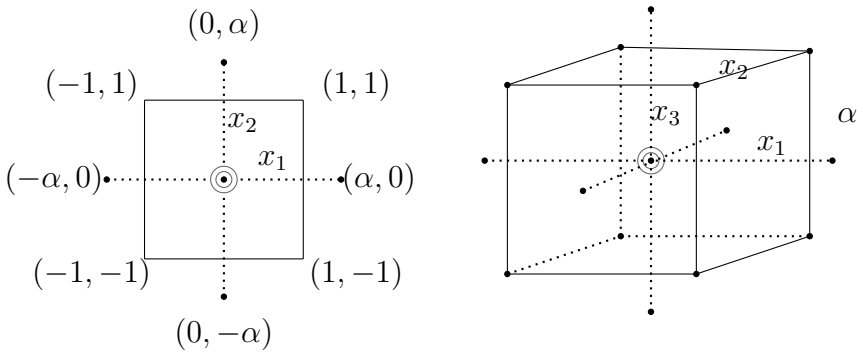


Figure 2.2: CCD for $m = 2$ and $m = 3$

Central composite designs (CCDs)

Introduced by Box and Wilson (1951) as an extension of factorial designs, they were developed as a way to efficiently fit quadratic response surfaces and identify optimal process settings in industrial experiments. CCDs are full or fractional factorial designs that are augmented with two additional sets of sampling points described as "center" and "axial or star" points (Box and Wilson 1951). The center point is defined by all factors being set at their center level. The CCD uses $2m$ axial points, each of which is defined by all but one factor being at their center level and the level of the remaining factor is denoted by α, which is generally chosen to be between 1 and $\sqrt{m}$ (Montgomery 2017). The basic concepts of the CCD for $m = 2$ and $m = 3$ are depicted in Figure 2.2, where the bold dots are the design points.

Orthogonal array composite designs (OACD)

Introduced by Xu, Jaynes, and Ding (2014), an OACD is a class of composite designs based on a two-level factorial design and a three-level orthogonal array (OA). An OA of n runs, m columns, s levels, and strength t, denoted by $\mathrm{OA}(n, s^m, t)$, is an $n \times m$ matrix in which all s^t level-combinations appear equally often in every $n \times t$ submatrix (Wu and Hamada 2011). For example, a 2^m factorial design can be viewed as $\mathrm{OA}(n, 2^m, t)$ with $n = 2^m$ and $t = m$.

Similarly, a three-level OA can be written as OA $(n, 3^m, t)$. Thus, an OACD is a composite design which consists of a two-level factorial design as its factorial points, a three-level OA as its additional points, plus any number of center points (Luna et al. 2022).

Space filling designs

While the previously discussed designs utilize sampling points that are at the boundaries of the design space, space filling designs generate samples that are dispersed throughout the multidimensional design space and not just at the boundary of the design space. These designs are important in sampling a surface as they could capture important regions thereby minimizing the bias between the true structure of the surface and the estimated surface from the sampled points (Gardner et al. 2006; Giunta, Wojtkiewicz, and Eldred 2003). They are of various types depending on the approach used to construct them, e.g., sampling-based Latin Hypercube designs, distance-based maximin designs, and distribution based uniform designs (Burton, Xu, and Yi 2022).

Latin hypercube designs (LHDs) – Based on McKay (1992)'s Latin hypercube sampling, it divides the range of each factor into bins of equal size, where n also corresponds to the number of samples to be generated resulting in a total of n^m combinations where m is the number of factors being considered. The n samples are then randomly generated such that for all one-dimensional projections, there will be only one sample in each bin.

Maximin distance designs – Introduced by Johnson, Moore, and Ylvisaker (1990), this design aims at spreading out the design points in the design space by maximizing the minimum distance between any two design points. It thus tends to place a large proportion of points at the corners and on the boundaries of the design space. Mathematically, this can be formulated as follows. Suppose we want to construct an n-run design in m factors. Let the design region be the unit hypercube $\mathcal{X}$ and let the design be $D = \{\boldsymbol{x}_1 \ldots, \boldsymbol{x}_n\}$, where each design point $\boldsymbol{x}_i$ is in $\mathcal{X} = [0, 1]^m$. The maximin design

optimizes the function below:

$$\max_{D} \min_{i \neq j} d\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right),$$

where $d\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)$ is the distance between the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

Maximin Latin hypercube designs – Unlike Latin hypercube designs, maximin distance designs do not have good projection properties for each factor. Morris and Mitchell (1995) proposed to overcome this problem by searching for the maximin distance design within the class of Latin hypercube designs. They also proposed to use the following criterion to achieve maximin distance:

$$\min_{D} \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{d^p\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)} \right\}^{1/p} \tag{2.3}$$

where $p > 0$ is chosen large enough, say $p = 15$.

Maximum projection (MaxPro) designs – Although maximin Latin hypercube designs ensure good space-filling in m dimensions and uniform projections in each dimension, their projection properties in two to $m - 1$ dimensions are not known. By the effect sparsity principle (Wu and Hamada 2011), only a few factors are expected to be important. To curb this, Joseph, Gul, and Ba (2015) proposed a different criterion:

$$\min_{D} \psi(D) = \left\{ \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{\prod_{l=1}^{m} \left(x_{il} - x_{jl}\right)^2} \right\}^{1/m}. \tag{2.4}$$

and showed that the design that minimizes $\psi(D)$ tends to maximize its projection capability in all subspaces of factors, and thus named these designs as maximum projection designs.

2.4 Modeling

We consider three different frameworks to model the data: (a) Linear Model, (b) Kriging Model and (c) Heterogeneous Gaussian Process (HetGP).

Linear Model (lm)

For m quantitative factors, denoted by $x_1, \ldots, x_m$, a second-order linear model is defined as

$$y = \beta_0 + \sum_{i=1}^{m} \beta_i x_i + \sum_{i=1}^{m} \beta_{ii} x_i^2 + \sum_{i<j} \beta_{ij} x_i x_j + \epsilon \tag{2.5}$$

where $\beta_0, \beta_i, \beta_{ii}$, and β_{ij} are the intercept, linear, quadratic, and bilinear (or interaction) terms, respectively, and ϵ is the error term. This model is simple and provides a straightforward way to model and understand relationships between the response variable and the factors. The main effects are easy to to interpret.

Kriging Model (km)

Proposed by South African geostatistician Krige (1951), Kriging is one of the methods used to interpolate intermediate values, whereby these intermediate values are modeled using Gaussian Process (GP) which is governed by prior co-variances. It provides a probabilistic prediction of the output variable, as well as an estimate of the uncertainty of the prediction (Chevalier, Picheny, and Ginsbourger 2014). The kriging predictors interpolating the observations are assumed to be noise-free (Roustant, Ginsbourger, and Deville 2012). Intermediate interpolated values obtained by Kriging are the best linear unbiased predictors.

The Kriging model consists of two parts: a trend and a GP. The trend part is often modeled as a regression on some fixed basis functions. In the specific case where the basis functions reduce to a constant function, it is referred to as ordinary Kriging (Roustant, Ginsbourger, and Deville 2012). Here is the general form:

$$Y(\boldsymbol{x}) = \sum_{i=1}^{k} \beta_i f_i(\boldsymbol{x}) + Z(\boldsymbol{x}), \tag{2.6}$$

where $f_1, \ldots, f_k$ are k basis functions, $\beta_1, \ldots, \beta_k$ are corresponding regression coefficients, and $Z(\boldsymbol{x})$ is a stationary GP with zero mean and covariance function ψ. The covariance

function ψ completely defines the behavior of the Gaussian Process $Z(\boldsymbol{x})$. It is defined as:

$$\psi\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \operatorname{Cov}\left(Z\left(\boldsymbol{x}_i\right), Z\left(\boldsymbol{x}_j\right)\right) = \sigma^2 \prod_{l=1}^{m} K\left(h_l; \theta_l\right), \tag{2.7}$$

where σ^2 is the scale parameter called the process variance, $h_l = |x_{i,l} - x_{j,l}|$, $x_{i,l}$ and $x_{j,l}$ are the lth elements of the i^{th} run $\boldsymbol{x}_i$ and the j^{th} run $\boldsymbol{x}_j$, and $K(h;\theta)$ is the correlation function. The parameter θ_l chosen for the correlation function $K\left(h_l;\theta_l\right)$ must be positive. Otherwise the correlation function will not be feasible. The parameters θ_l are chosen to be physically interpretable in the same unit as the corresponding variables. They are often referred to as the *characteristic length-scales* by Rasmussen and Williams (2006). Popular correlation functions include Gaussian, Matérn, and power-exponential family correlation functions. The Matérn function with parameter $\nu = 5/2$ is often chosen as the default when fitting kriging models. It is defined as:

$$K(h;\theta) = \left(1 + \frac{\sqrt{5}h}{\theta} + \frac{5h^2}{3\theta^2}\right) \exp\left(-\frac{\sqrt{5}h}{\theta}\right). \tag{2.8}$$

The unknown parameters can be estimated via MLE or cross validation. We use the function `km` in the R package DiceKriging to fit this kind of model.

Heteroskedastic Gaussian Process (HetGP)

HetGP follows the simplifying assumption in the computer experiments literature in using a mean zero GP, which shifts all of the modeling effort to the covariance structure (Binois, Gramacy, and Ludkovski 2018). Observation model is given by

$$y_i = y\left(\boldsymbol{x}_i\right) = f\left(\boldsymbol{x}_i\right) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}\left(0, r\left(\boldsymbol{x}_i\right)\right), \tag{2.9}$$

where $f(\boldsymbol{x}_i)$ is a GP with covariance or kernel $k(\cdot,\cdot)$ and $r(\boldsymbol{x}_i)$ is the variance of ϵ_i which depends on $\boldsymbol{x}_i$. The kernel $k(\cdot,\cdot)$ is positive definite, with parameterized families such as the Gaussian or Matérn being typical choices. If $r(\boldsymbol{x}_i)$ is a constant, then the process is homoskedastic. In matrix notation, the modeling framework just described is equivalent to

writing

$$Y \sim \mathcal{N}\left(\mathbf{0}, \mathbf{K}_n + \mathbf{\Sigma}_n\right),$$

where $\mathbf{K}_n$ is the $n \times n$ matrix with (i, j) coordinate $k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)$, and $\mathbf{\Sigma}_n = \text{Diag}\left(r\left(\boldsymbol{x}_1\right), \ldots, r\left(\boldsymbol{x}_n\right)\right)$ is the variance matrix of the vector of independent noise ε_i.

Given the kernel function $k(\cdot, \cdot)$ and data $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$, multivariate normal (MVN) conditional identities provide a predictive distribution at site $\boldsymbol{x} : Y(\boldsymbol{x}) \mid \boldsymbol{y}$, which is Gaussian with parameters

$$\mu(\boldsymbol{x}) = \mathbb{E}(Y(\boldsymbol{x}) \mid \boldsymbol{y}) = \mathbf{k}(\boldsymbol{x})^\top \left(\mathbf{K}_n + \mathbf{\Sigma}_n\right)^{-1} \boldsymbol{y},$$

$$\sigma^2(\boldsymbol{x}) = \mathbb{V}\,\text{ar}(Y(\boldsymbol{x}) \mid \boldsymbol{y}) = k(\boldsymbol{x}, \boldsymbol{x}) + r(\boldsymbol{x}) - \mathbf{k}(\boldsymbol{x})^\top \left(\mathbf{K}_n + \mathbf{\Sigma}_n\right)^{-1} \mathbf{k}(\boldsymbol{x}),$$

where $\mathbf{k}(\boldsymbol{x}) = \left(k\left(\boldsymbol{x}, \boldsymbol{x}_1\right), \ldots, k\left(\boldsymbol{x}, \boldsymbol{x}_n\right)\right)^\top$. We use the function `mleHetGP` in the R package hetGP to fit this model with the default Gaussian kernel.

2.5 The Data Generation Process

We aim to obtain optimal DE hyperparameter settings that can be used to generate UPDs.

The Objective Function: Uniform Projection Design Criterion

Proposed by Sun, Wang, and Xu (2019), the uniform projection criterion solely focuses on two-dimensional projections. This is due to two factor interactions being more important than three-factor or higher-order interactions. The motivation idea was that although designs with low discrepancy have good uniformity in the full-dimensional space, they can have bad projections in lower dimensional spaces, which is undesirable when only a few factors are active. Thus we prefer designs with better projection properties. Sun, Wang, and Xu (2019) argued that the uniform projection designs scatter points uniformly in all dimensions and have good space-filling properties in terms of distance, uniformity and orthogonality.

The uniform projection design criterion is defined using the centered L_2-discrepancy. For an $n \times m$ design $D = (x_{ik})$ with s levels from $\{0, 1, \ldots, s-1\}$, its (squared) centered

L_2-discrepancy is defined as

$$\begin{aligned}\mathrm{CD}(D) =& \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\prod_{k=1}^{m}\left(1+\frac{1}{2}\left|z_{ik}\right|+\frac{1}{2}\left|z_{jk}\right|-\frac{1}{2}\left|z_{ik}-z_{jk}\right|\right)\\ &-\frac{2}{n}\sum_{i=1}^{n}\prod_{k=1}^{m}\left(1+\frac{1}{2}\left|z_{ik}\right|-\frac{1}{2}\left|z_{ik}\right|^2\right)+\left(\frac{13}{12}\right)^m,\end{aligned}$$

where $z_{ik} = (2x_{ik} - s + 1)/(2s)$. Then the uniform projection criterion is to minimize

$$\phi(D) = \frac{2}{m(m-1)}\sum_{|u|=2}\mathrm{CD}\left(D_u\right), \tag{2.10}$$

where u is a subset of $\{1, 2, \ldots, m\}$, $|u|$ denotes the cardinality of u and D_u is the projected design of D onto dimensions indexed by the elements of u. The $\phi(D)$ is the average centered L_2-discrepancy values of all two-dimensional projections of D.

We implemented the DE and uniform projection criterion in the package `UniPro`. The following code generates an $n \times m$ UPD with s levels

```
> UniPro(n, m, s, NP, itermax, pMut, pCR, pGBest, seed)
```

where NP, itermax, pMut, pCR and pGBest are DE hyperparameters described in Section 2, and seed is an optional seed for random number generators.

As the task of design generation is quite complex, only 3 design sizes are considered. A UPD of size 30×3 is considered as a small and easy task, 50×5 as a medium task and 70×7 as a large and difficult task. We only consider the construction of designs with $s = n$ so that the resulting UPD is an LHD.

Training and testing data

Designs discussed in Section 2.3 are used to determine the parameter settings for the DE algorithm hyperparameters. Specifically, we construct five designs: a CCD with 43 runs (`ccd3_43`), an OACD with 50 runs (`oacd3_50`), 50-run random LHD, 50-run maximin LHD, and 50-run maxpro LHD. All designs have five factors, one for each hyperparameter. Each

run corresponds to a setting of the five hyperparameters. The CCD and OACD have 3 levels while the three LHDs have 50 levels. The levels are linearly interpolated within the minimum and maximum factor values for each hyperparameter.

Given the target design size ($n \times m$) and a setting of the five hyperparameters, we ran the `UniPro` function to generate an $n \times m$ UPD and the resulting $\phi(D)$ value defined in (2.10). This is recorded as the response value for that particular hyperparameter setting and target design size. For each setting, the DE algorithm is replicated ten times yielding ten replicates for the response. These are then aggregated to obtain the mean and the standard deviation of the response. Thus we obtain five training datasets for each target size.

The same procedure is taken to generate the testing dataset with the exception that the designs used for the hyperparameter setting combinations being a 3^5 full factorial design, a random LHD with 243 runs, a combination of these two, and a 4^5 full factorial design.

Density plots of the response for the testing and training data are presented in Figure 2.3 for the target size 50×5. All of the distributions are skewed to the right. All designs lead to similar minimum $\phi(.)$ values, around 0.17, whereas different types of designs lead to different maximum $\phi(.)$ values. Indeed, all space filling designs have maximum $\phi(.)$ values around 0.28, while the factorial designs and the hybrid design have a maximum $\phi(.)$ values around 0.34. The narrower range of the $\phi(.)$ values suggests that the space filling designs do not explore the entire space of hyperparameters.

Model evaluation

For each training dataset, we fit the three models descibed in Section 2.4 and test on the four testing datasets. Designs are evaluated by considering their ability to collect informative data for building a statistical model that specifies the relationship between the response and the hyperparameters, which is measured by the test root mean squared error (RMSE). The correlation (ρ) between the response and the predicted together with the RMSE are reported. Generating the test data from the 3^5 factorial, 243-run random LHD, and the 4^5

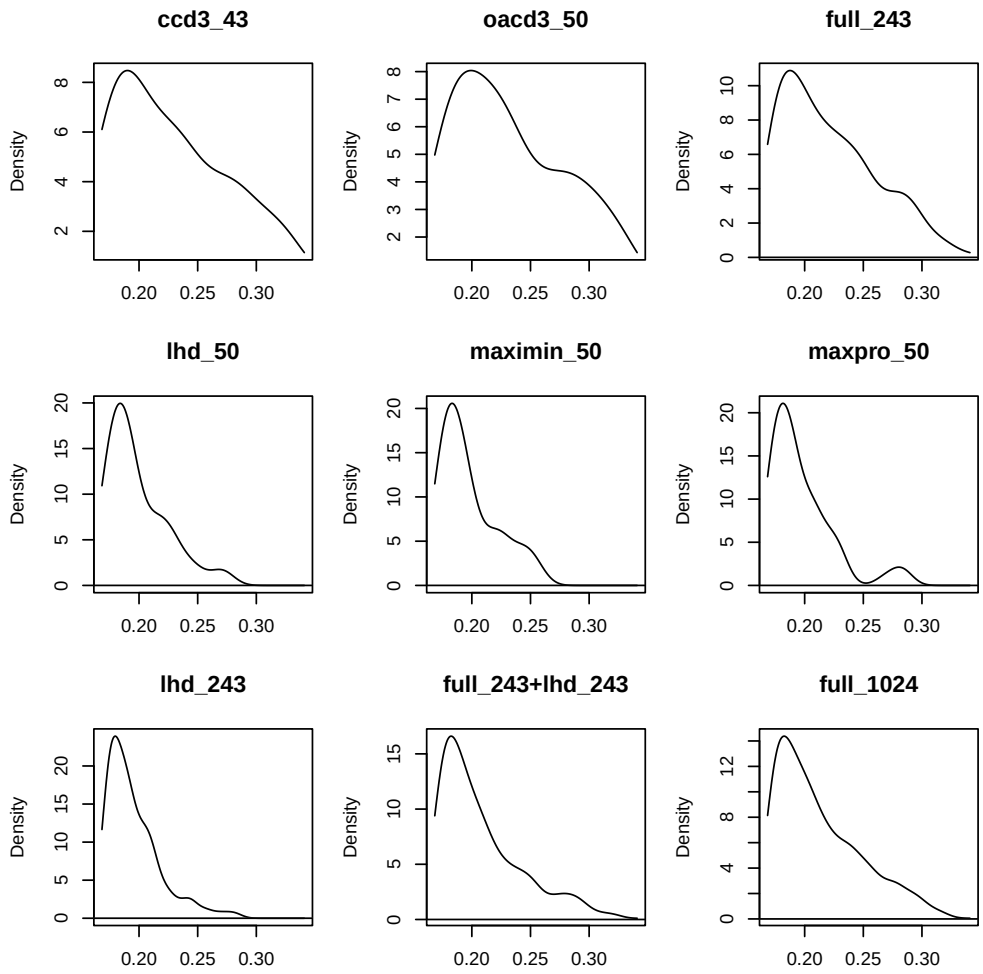


Figure 2.3: Density plots of the $\phi(D)$ values with target size 50×5

Table 2.1: Comparison of designs and model evaluations with target size 30×3

	(a) Testing on the 3^5 FFD						**(b) Testing on the 243 LHD**					
	correlation			RMSE			correlation			RMSE		
Design	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP
ccd3_43	0.88	0.88	0.88	1.51	1.59	1.62	0.65	0.03	0.60	1.62	3.75	1.51
oacd3_50	0.88	0.94	0.93	1.62	1.25	1.32	0.68	0.63	0.61	1.94	2.30	2.22
lhd_50	0.41	0.36	0.34	3.19	3.30	3.27	0.28	0.20	0.19	1.08	1.14	1.14
maximin_50	0.71	0.73	0.74	2.86	3.19	3.03	0.64	0.63	0.64	0.66	0.66	0.65
maxpro_50	0.71	0.62	0.66	2.35	2.97	2.72	0.69	0.71	0.74	0.74	0.69	0.61
	(c) Testing on the 3^5 FFD+243 LHD						**(d) Testing on the 4^5 FFD**					
	correlation			RMSE			correlation			RMSE		
Design	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP
ccd3_43	0.84	0.60	0.83	1.57	2.88	1.57	0.84	0.66	0.85	1.55	2.83	1.50
oacd3_50	0.82	0.83	0.83	1.79	1.85	1.83	0.85	0.87	0.87	1.68	1.68	1.68
lhd_50	0.42	0.35	0.36	2.38	2.47	2.45	0.45	0.39	0.37	2.47	2.57	2.56
maximin_50	0.69	0.63	0.68	2.08	2.31	2.19	0.74	0.75	0.76	2.16	2.38	2.27
maxpro_50	0.74	0.60	0.68	1.75	2.15	1.98	0.74	0.67	0.71	1.80	2.22	2.03

factorial seems good enough. This is because the random LHD enjoys the maximum space-filling property in all one dimensions, while the 3^5 and 4^5 factorial designs cover the entire 5-dimensional input space in a uniform fashion.

2.6 Results and Analysis

Table 2.1(a)(b) and Figure 2.4(a)(b) present comparison of designs and model evaluations with target size 30×3 for testing the two 243-run data sets. One striking observation is

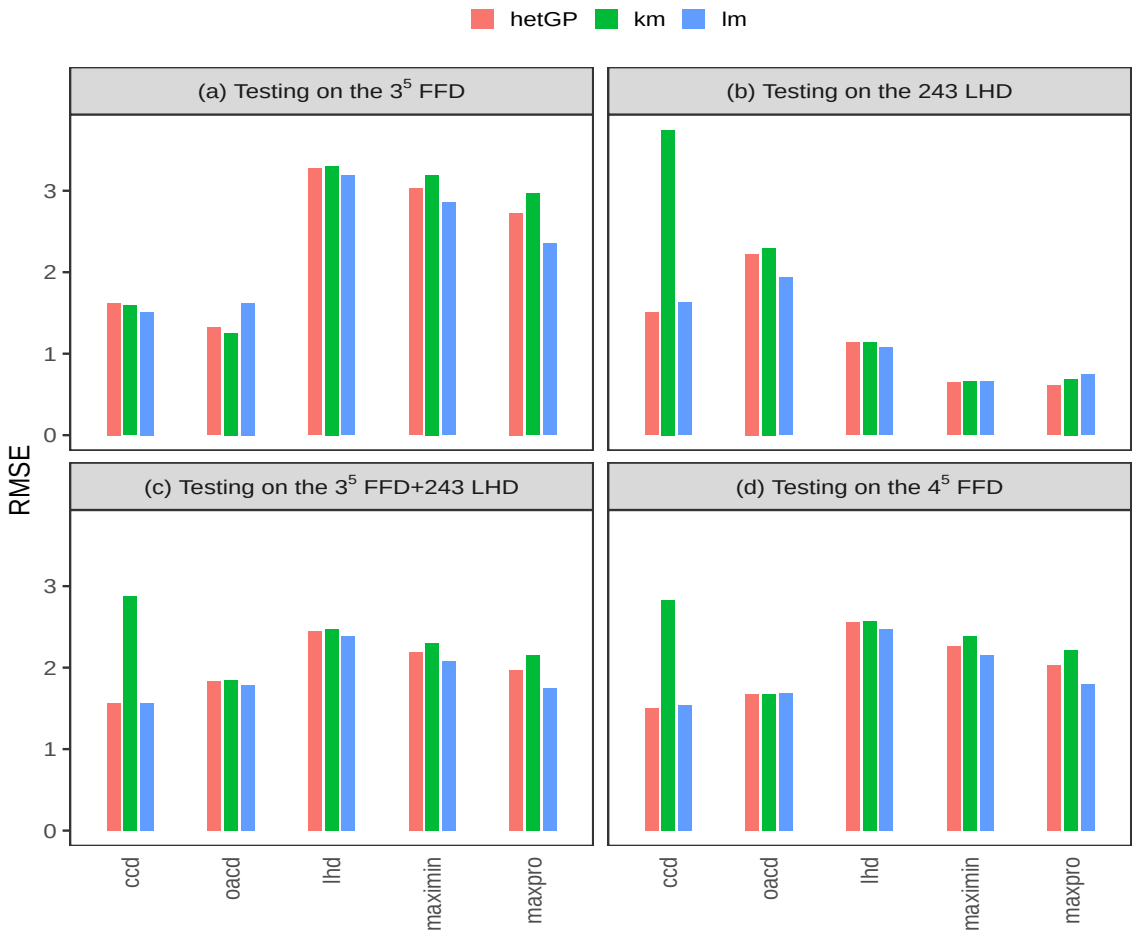


Figure 2.4: Comparison of RMSE with target size 30×3

that the performance of the training data set depends on the nature of the testing data set. The composite designs, CCD and OACD, seem to be better when tested on the 3^5 FFD while the space filling designs (random LHD, maximin LHD, and maxpro LHD) did better when tested on the 243-run random LHD. As this does not give a general idea as to which designs might perform better in general, we invoke the combined data with 486 runs and the 4^5 FFD as the testing dataset. Here we see that the composite designs perform better than the space filling designs; see Table 2.1(c)(d) and Figure 2.4(c)(d). The 50-run random LHD performed the worst in terms of correlation regardless of the testing data. The correlation is strikingly low whereas the RMSE is high. This might be due to randomness, but it does show the weakness of the random LHD.

One other bizarre observation from Table 2.1(b) is the correlation of 0.03 when using the CCD as the training design and testing it on the 243-run random LHD. This value is strikingly lower than any other values given in Table 2.1. No apparent reason could be deduced as to why this is so. Multiple replications indicated that this is not an error. From all the results, we can deduce the robustness of OACD over CCD. This gives a reason to use OACD for hyperparameter initialization.

With regards to the models, there seems to be no striking observation to be made as to whether one fitting method performs better than the other two, with exception for one 30×3 case when the kriging model fitting to the CCD training data had a much higher RMSE value than the other cases.

The three models have quite different assumptions. The linear model assumes a polynomial trend and independent random errors with homoskedastic variance. The Kriging model assumes a stationary covariance structure, that is, the covariance between two points in the DE hyperparameter surface structure depends only on the distance or spatial lag between those points, and not on their specific locations within the domain. The HetGP model assumes a heteroskedastic variance-covariance structure. For the DE algorithm, the homoskedastic and stationary assumptions are questionable. Due to this, the HetGP model

is preferred to the linear model and the kriging model. However, the linear model is easy to interpret and fits as well as the HetGP model. We will use the linear model to determine factor importance and optimal hyperparameter settings in the next section.

Tables 2.3-2.4 and Figures 2.9-2.10 in the Appendix show results when the target design sizes are 50×5 and 70×7, respectively. Looking at the results, apart from the random LHD with 50 runs, previously stated observations are affirmed. The composite designs perform better when tested on the 3^5 FFD while the space filling designs perform better when tested on the 243-run LHD. When tested on the 486-run combined data and the 4^5 FFD, the composite designs perform better than the space filling designs, although the difference tends to be small as the values are very close. In addition, there is no clear difference between the performances of the three models.

A natural question is why composite designs perform better than the space filling designs. We perceive that the hyperparameters at the boundaries lead to some extreme cases in this experiment and the composite designs do capture this phenomena while the space filling designs do not. Figure 2.5 presents the histograms of the distances from design points to the design center for all the designs, where each column is rescaled to $[-1, 1]$ and the Euclidean distance from each point to the center of the design is calculated. This gives an insight as to why the composite designs, OACD and CCD, tend to perform better than the space filling designs. This is because the composite designs tend to capture information lying at the boundaries compared to the space filling designs which tend to capture the information lying at the center of the design. This is confirmed by the notion that three of the hyperparameters tend to be optimized around their highest level as discussed in the next section.

2.7 Factor Importance and Optimal Settings

The results obtain call for a deeper look into the models and how each factor is involved in the surface approximation. This enables us to have a better picture of the surface generated by the DE hyperparameters.

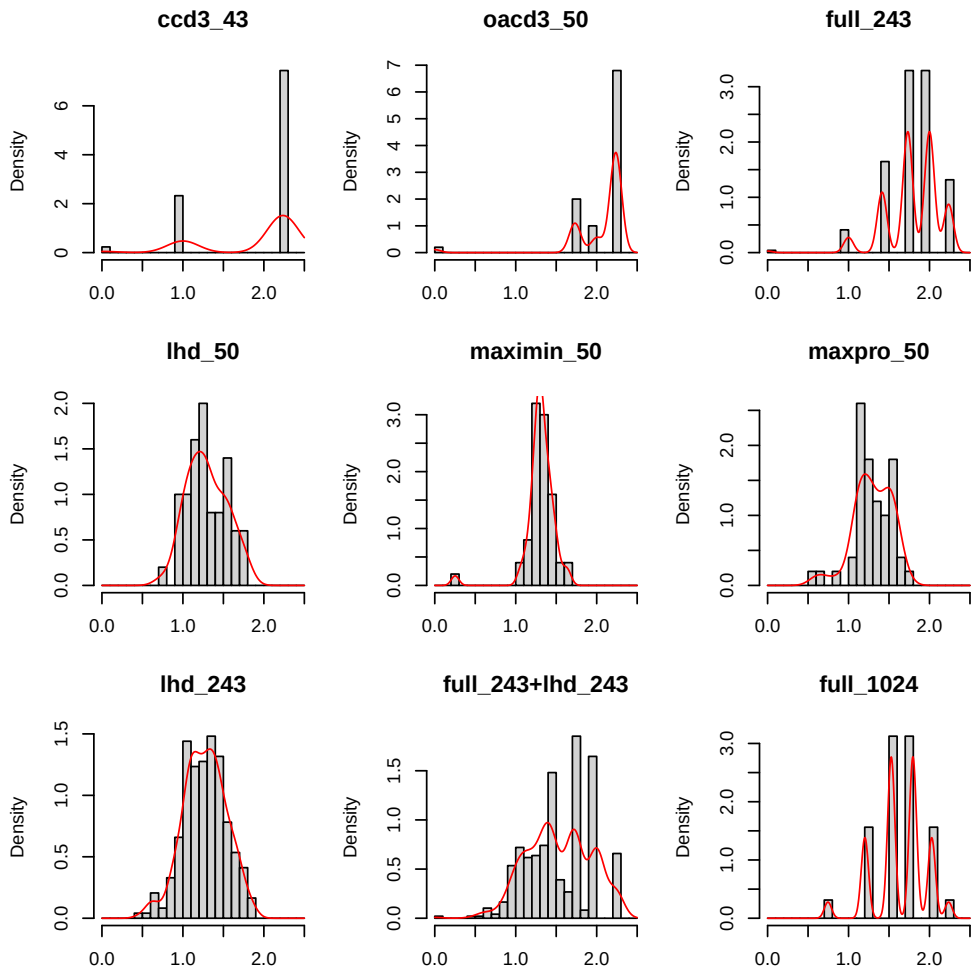


Figure 2.5: Histogram of the distances from design points to the design center

	ccd3_43	oacd3_50	maximin_50	maxpro_50
(Intercept)	0.3815^{***}	0.3876^{***}	0.3878^{***}	0.3814^{***}
NP	-0.0134^{***}	-0.0143^{***}	-0.0042^{***}	-0.0055^{*}
pMut	0.0020	0.0028	0.0043^{***}	0.0045^{*}
pGBest	-0.0204^{***}	-0.0224^{***}	-0.0047^{***}	-0.0082^{***}
pCR	−0.0022	−0.0030	0.0004	0.0007
itermax	-0.0146^{***}	-0.0152^{***}	-0.0046^{***}	−0.0021
itermax_q	0.0090	−0.0081	0.0011	0.0057
NP_q	0.0119	0.0080	0.0022	−0.0004
pCR_q	0.0071	0.0074	−0.0008	0.0049
pGBest_q	0.0083	0.0192^{*}	0.0035	0.0159^{***}
pMut_q	0.0150	0.0174^{*}	0.0083^{***}	0.0104^{*}
NP:pMut	0.0058	0.0064^{*}	−0.0002	−0.0037
NP:pGBest	−0.0041	−0.0038	0.0017	0.0006
NP:pCR	0.0002	−0.0007	−0.0006	−0.0002
NP:itermax	0.0049	0.0033	0.0023	0.0052
pMut:pGBest	-0.0080^{*}	-0.0093^{***}	-0.0089^{***}	-0.0139^{**}
pMut:pCR	0.0203^{***}	0.0197^{***}	0.0042^{**}	0.0018
pMut:itermax	0.0032	0.0025	-0.0061^{***}	0.0004
pGBest:pCR	0.0034	0.0036	0.0000	0.0051
pGBest:itermax	0.0057	0.0068^{**}	0.0076^{***}	0.0058
pCR:itermax	0.0037	0.0021	0.0018	0.0040
R^2	0.9034	0.9235	0.8970	0.7297
Adj. R^2	0.8157	0.8708	0.8260	0.5433
Num. obs.	43	50	50	50

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

Table 2.2: Statistical models

Table 2.2 shows the estimated coefficients of the second-order models based on the four different training datasets: CCD, OACD, maximin LHD, and maxpro LHD, for the target size 30×3. Looking at the various models above, the model obtained from using the maxpro LHD training data is the worst performing. It has the lowest adjusted R^2 of only 0.54. This model does not capture important main effects. For example, it indicates that the number of iterations (itermax) for optimization is not significant. Yet this is well known to be important. On the other hand, the model obtained by using OACD performs the best. It has an adjusted R^2 of 0.87 and captures important main effects and interactions. In addition, CCD might be a little worse than OACD because of the fewer number of points (43) used for training compared to the other models which used 50 points. The model from the CCD does not identify any of the quadratic effects to be significant while the other models do.

Looking at the corresponding p-values from the models obtained, it is evident that all five hyperparameters are important. The main effects of three hyperparameters (NP, itermax and pGBest) are very significant, whereas the several interactions involving one of the other two hyperparameters (pMut and pCR) are also very significant. The probability of using the global best (pGBest) is quite important because its main effect is highly significant in all the models in Table 2.2. It can also be inferred that the maximum number of iterations (itermax) and the population size (NP) are important. The linear models indicate that to minimize the objective function, it is ideal to use a larger pGBest, increase the population size (NP) and increase the number of iterations (itermax). The population size and the number of iterations could be limited by the available budget.

With regards to interactions, all significant interaction terms involve either pMut or pGBest. The interaction of pMut and pGBest is negative indicating an inverse relationship between the probability of mutation and probability of using the global best while the interaction between pMut and pCR is positive.

To have a better understanding of the interactions, we examine some interaction plots from the full 4^5 factorial, shown in Figure 2.6. From the interaction plots, it is advisable

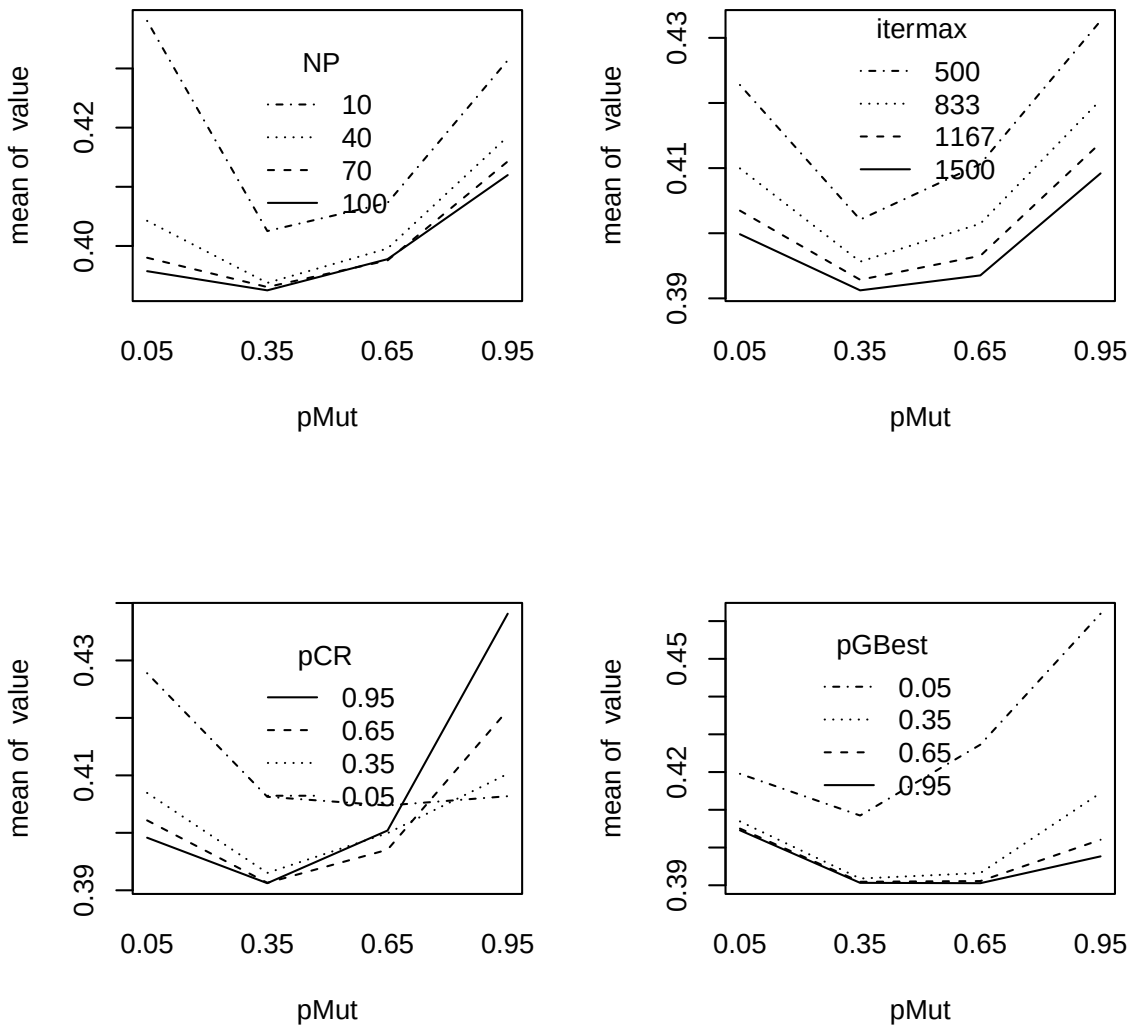


Figure 2.6: Interaction plots involving pMut based on the 4^5 FFD and target size 30×3.

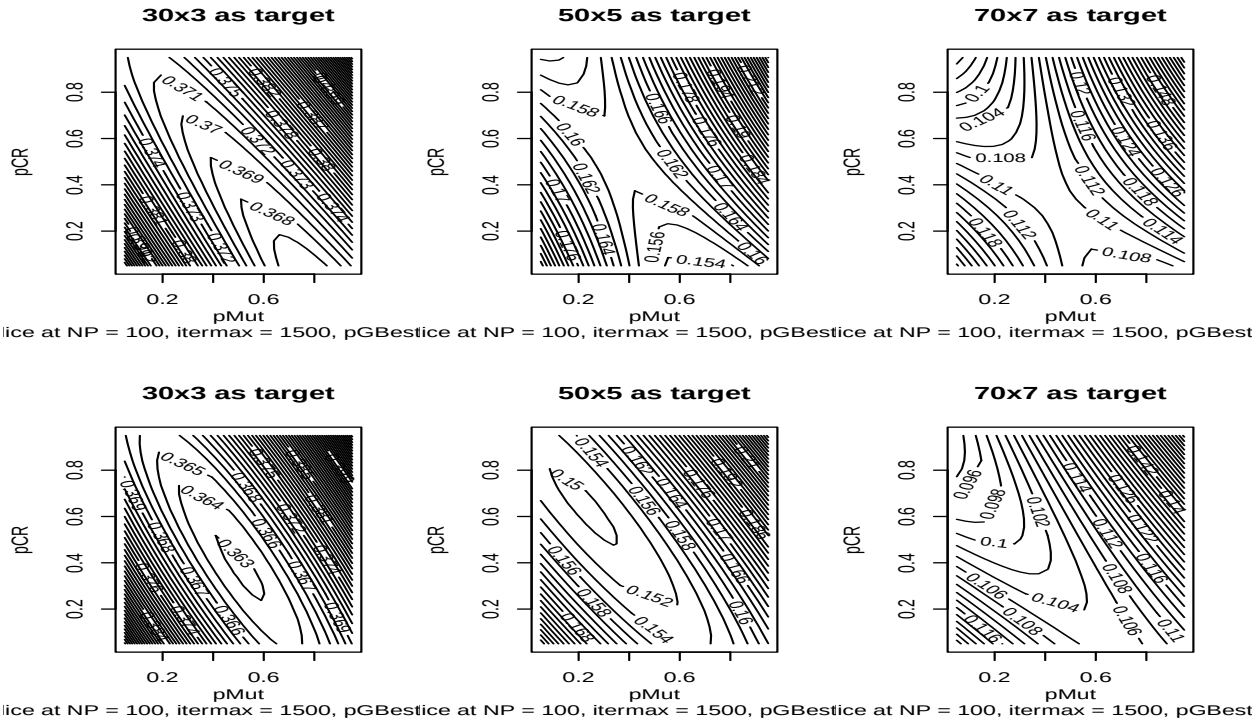


Figure 2.7: Contour plots of pMut and pCR while fixing other hyperparameters at high levels. Top row uses CCD as the training data; bottom row uses OACD as training data.

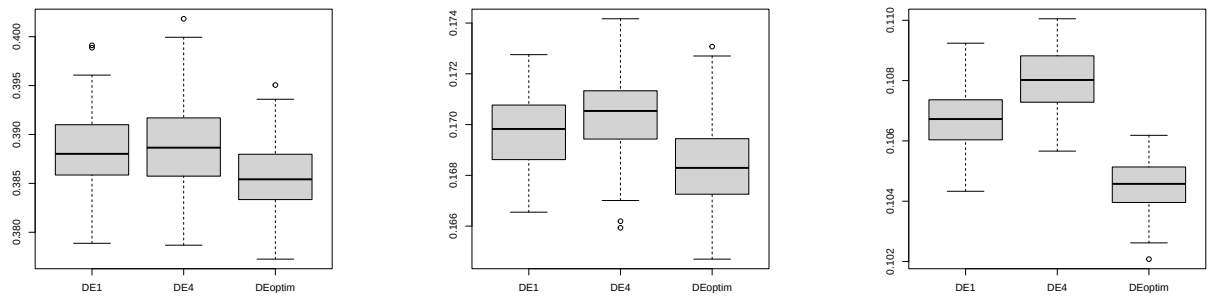
to set the number of population (NP), number of iterations (itermax) and the probability of using global best (pGBest) at their highest levels, i.e., 100, 1500 and 0.95, respectively. These results are consistent with the second order models in Table 2.2, where NP, itermax and pGBest are all significant and negative. Thus they should be set at the highest level to minimize the response values. These founding are consistent for other target design sizes. On the other hand, the interaction of the pMut and pCR is more complicated and a further analysis is called for to determine the preferred levels to set these parameters.

Figure 2.7 shows the contour plots of pMut and pCR while fixing NP=100, itermax=1500 and pGBest=0.95. From the contour plot based on CCD and target size 30×3, we note that the response value could be minimized by taking a small pCR and a large pMut. Whereas, in the cases of target sizes 50×5 and 70×7, the minimum could occur at the top left corner or the bottom right corner, which indicates that we could use a large pCR combining with a

small pMut or a small pCR combining with a large pMut. However, the contour plots based on OACD provide different recommendations. These results reveal that it is unlike to have a simple generic setting for pMut and pCR because these two parameters depend highly on the training dataset and also on the size of the target design. A further study focusing on pCR and pMut is needed to determine their optimal settings.

Here we take a simple approach. Figure 2.7 indicates that the good settings are along the diagonal line connecting the upper left corner and the lower right corner. So we search the optimal setting by simply varing pMut from $0.05, 0.15, \ldots, 0.95$ and taking pCR=1-pMut. For each setting of pMut and pCR, we run the DE algorithm 100 times to construct 100 designs while fixing NP=100, itermax=1500 and pGBest=0.95. We obtain the average response values and determine the optimal setting of pMut and pCR for each target size. This leads to the folllowing optimal settings of (pMut, pCR): (0.95, 0.05), (0.25, 0.75), and (0.15, 0.85), respectively, for target size 30×3, 50×5 and 70×7.

We compare the optimal settings with two DE variants (DE1 and DE4) provided by Stokes, Wong, and Xu (2024). DE1 always uses the global best (i.e., pGBest=1), while DE4 is a hybrid version with pGBest=0.5, which randomly chooses the global best, the current agent and a random agent with probability 50%, 25%, and 25% for each column independently. Both DE1 and DE4 use pMut = 0.1 and pCR = 0.5. For comparison purposes, we set pMut and pCR according to the optimal settings obtained above for each target size, while keeping pGBest=0.95. For each case, we set NP=100 and itermax=1500, and run DE algorithm 100 times to construct 100 designs for each method. Figure 2.8 shows boxplots of the $\phi(\cdot)$ values of these designs. It is evident that the DE with optimal settings (DEoptim) yields significantly better uniform projection designs than two settings provided by Stokes, Wong, and Xu (2024). The results confirm the efficiency of the hyperparameter tuning.



(a) 30×3 as target (b) 50×5 as target (c) 70×7 as target

Figure 2.8: Performance of the DE algorithms under three setttings: DE1, DE4 and DEoptim (optimal settings)

2.8 Conclusion

This paper compared small designs in exploring the response surface of a DE algorithm hyperparameters. We considered five numerical hyperparameters: population size, the number of maximum iterations, probability of crossover, probability of mutation, and probability of using the global best for mutation. Various composite designs and space-filling designs for selecting combinations of these hyperparameters were examined. We evaluated the performance of a design via building a second order model, a Kriging model and a heterogeneous GP model. The performance was measured in terms of testing RMSEs and correlation. The comparison was made based on data simulated using the uniform projection criterion. Under the settings we considered, the comparison demonstrates that OACD and CCD are the better choices than space-filling designs for exploring the response surface of the DE algorithm hyperparameters. In addition, the second-order model is simple and works as well as the Kriging model and the heterogeneous GP model. We demonstrated the importance of tuning the DE algorithm and provided a simple strategy on determining optimal hyperparameter settings for constructing UPDs with different target sizes.

While the primary goal of optimizing hyperparameters is to find an optimal hyperpa-

rameter combination that maximizes the overall performance of a learning algorithm, the present paper centers on examining the effect of various designs on the performance of hyperparameter tuning. This information is later used to determine the best hyperparamters that are then used by DE algorithm to generate UPDs. The process of obtaining the optimal hyperparameters, factor importance, whether the UPD generated is optimal and many more others are left out for future research.

Appendix: Additional tables and figures for target sizes 50×5 and 70×7

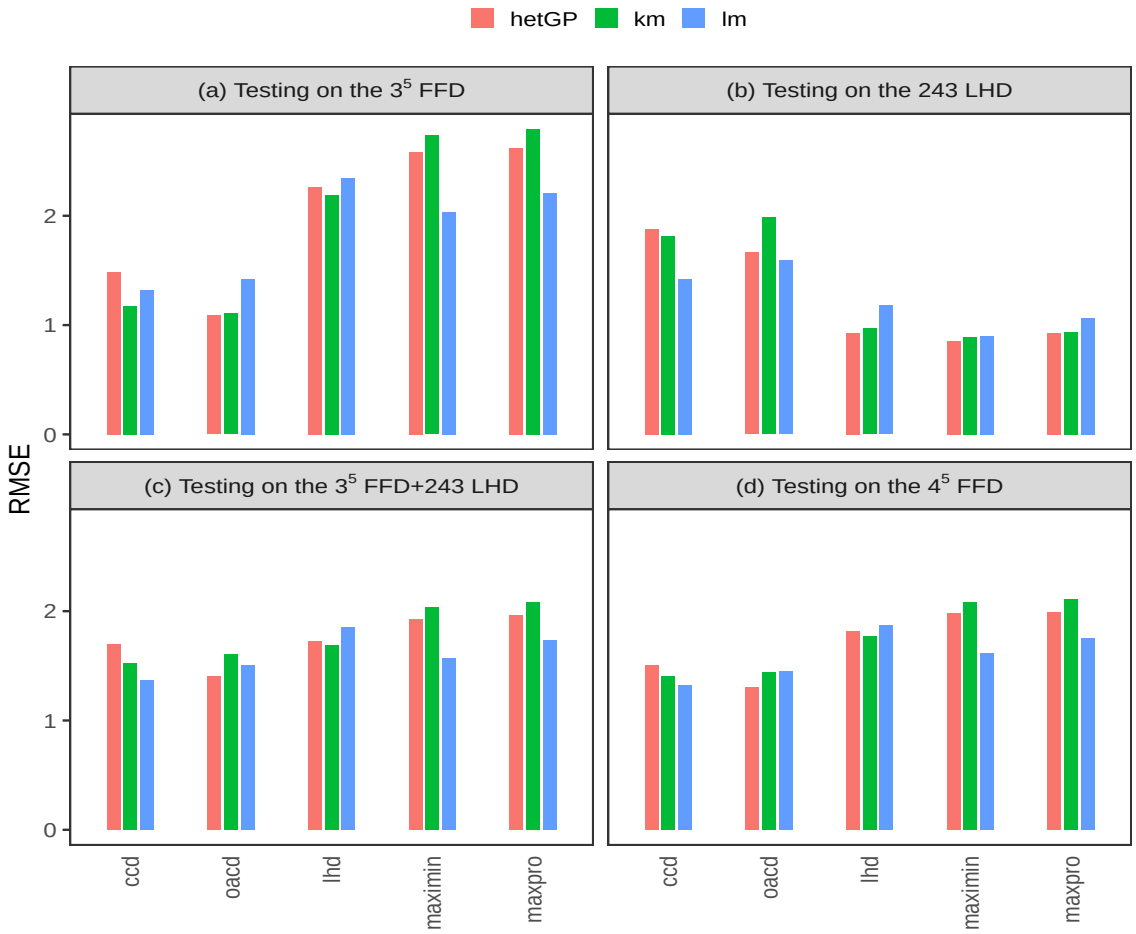


Figure 2.9: Comparison of RMSE with target size 50×5

Table 2.3: Comparison of designs and model evaluations with target size 50×5

	(a) Testing on the 3^5 FFD						**(b) Testing on the 243 LHD**					
	correlation			RMSE			correlation			RMSE		
Design	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP
ccd3_43	0.94	0.96	0.93	1.32	1.17	1.49	0.83	0.82	0.80	1.42	1.81	1.88
oacd3_50	0.94	0.96	0.96	1.42	1.11	1.09	0.83	0.84	0.86	1.59	1.98	1.66
lhd_50	0.83	0.85	0.83	2.35	2.19	2.26	0.89	0.92	0.93	1.18	0.97	0.92
maximin_50	0.87	0.84	0.81	2.03	2.73	2.58	0.92	0.92	0.93	0.90	0.89	0.85
maxpro_50	0.85	0.81	0.82	2.21	2.79	2.62	0.90	0.92	0.92	1.06	0.94	0.93
	(c) Testing on the 3^5 FFD+243 LHD						**(d) Testing on the 4^5 FFD**					
	correlation			RMSE			correlation			RMSE		
Design	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP
ccd3_43	0.92	0.92	0.89	1.37	1.53	1.70	0.93	0.93	0.92	1.32	1.40	1.50
oacd3_50	0.91	0.92	0.93	1.51	1.61	1.41	0.92	0.94	0.94	1.45	1.44	1.30
lhd_50	0.86	0.88	0.88	1.86	1.69	1.73	0.87	0.87	0.87	1.87	1.77	1.82
maximin_50	0.90	0.85	0.85	1.57	2.03	1.92	0.90	0.87	0.86	1.61	2.08	1.98
maxpro_50	0.88	0.83	0.85	1.73	2.08	1.97	0.88	0.85	0.86	1.75	2.11	1.99

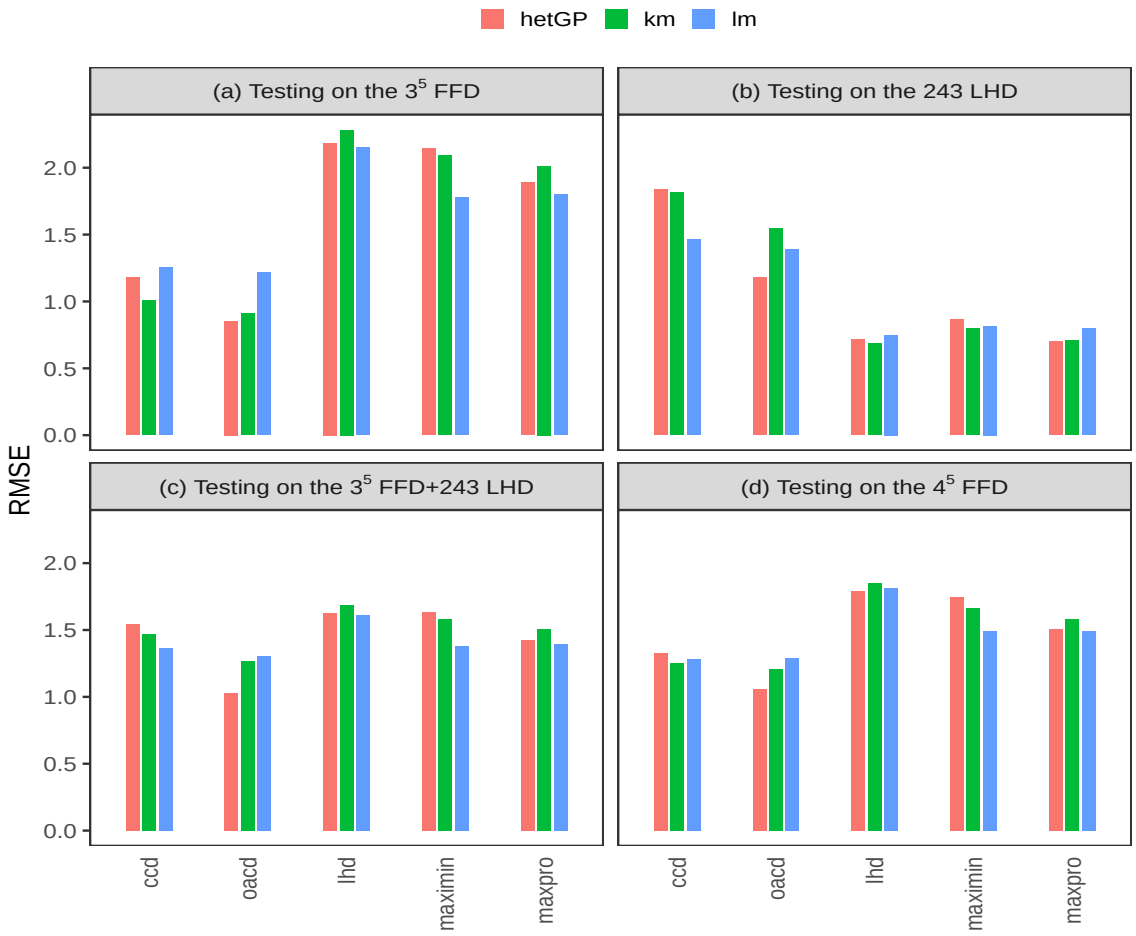


Figure 2.10: Comparison of RMSE with target size 70×7

Table 2.4: Comparison of designs and model evaluations with target size 70×7

	(a) Testing on the 3^5 FFD						**(b) Testing on the 243 LHD**					
	correlation			RMSE			correlation			RMSE		
Design	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP
ccd3_43	0.93	0.96	0.94	1.25	1.01	1.18	0.81	0.84	0.83	1.46	1.82	1.84
oacd3_50	0.94	0.97	0.97	1.22	0.91	0.86	0.83	0.88	0.92	1.39	1.54	1.18
lhd_50	0.83	0.84	0.82	2.15	2.28	2.18	0.94	0.95	0.95	0.75	0.69	0.72
maximin_50	0.87	0.87	0.81	1.77	2.09	2.14	0.94	0.94	0.93	0.82	0.80	0.86
maxpro_50	0.88	0.88	0.87	1.80	2.01	1.89	0.94	0.95	0.95	0.80	0.71	0.70
	(c) Testing on the 3^5 FFD+243 LHD						**(d) Testing on the 4^5 FFD**					
	correlation			RMSE			correlation			RMSE		
Design	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP	lm	km	hetGP
ccd3_43	0.90	0.91	0.90	1.36	1.47	1.54	0.92	0.94	0.93	1.28	1.26	1.33
oacd3_50	0.91	0.93	0.95	1.30	1.27	1.03	0.92	0.94	0.95	1.29	1.21	1.06
lhd_50	0.87	0.87	0.87	1.61	1.69	1.62	0.86	0.87	0.86	1.81	1.85	1.79
maximin_50	0.90	0.88	0.86	1.38	1.58	1.63	0.89	0.89	0.86	1.49	1.67	1.75
maxpro_50	0.90	0.89	0.90	1.39	1.51	1.43	0.90	0.90	0.90	1.49	1.58	1.51

CHAPTER 3

Evaluating Space-Filling Designs for Prediction and Sequential Optimization

3.1 Introduction

In optimization and experimental design, selecting the right strategy is key to achieving precise and efficient results, particularly in methods like Gaussian processes (GP) and Efficient Global Optimization (EGO). These techniques rely heavily on initial designs to guide optimization and improve predictive accuracy. Although the Latin hypercube design (LHD) (McKay, Beckman, and Conover 1979) is a commonly used space-filling strategy for computer experiments, research suggests that alternative designs may outperform it in certain cases, especially for screening and prediction tasks (Welch et al. 1992). Studies have explored various design strategies, finding that even subtle differences in design can significantly affect performance (Chen et al. 2016).

The choice of design is particularly important in complex systems, including fields like engineering and machine learning. Prior research has highlighted how design strategies can impact the performance of Gaussian processes, emphasizing the importance of selecting appropriate methods for high-dimensional problems (Harari and Steinberg 2014).

In a sequential optimization such as EGO, as optimization progresses, the GP model is updated with additional sample points, reducing the influence of the initial design. Nevertheless, the initial design strategy remains crucial in determining the speed and accuracy of convergence. Choosing the appropriate design strategy for a given problem is essential for

enhancing both predictive performance and optimization efficiency.

The objective of this paper is to evaluate which space-filling designs offer greater efficiency and robustness for both prediction and optimization. Various types of space-filling designs have been proposed in the literature. They include LHDs, maximin distance designs (Johnson, Moore, and Ylvisaker 1990; Morris and Mitchell 1995), orthogonal array-based designs (Tang 1993; Xiao and Xu 2018), uniform designs (Fang et al. 2000), maximum projection designs (Joseph, Gul, and Ba 2015), and uniform projection designs (Sun, Wang, and Xu 2019). Traditionally, comparisons have been focused on LHDs and maximin distance LHDs. With new software developments, we now compare the performance of other types of space-filling designs, including maximum projection designs, uniform designs, and uniform projection designs.

Recent studies, such as those by Shi, Chiu, and Xu (2023), have evaluated space-filling designs in deep neural network predictions, but our research extends this by focusing on their performance in optimization as well. We assess both predictive and minimization capabilities using deterministic test functions, with GP models serving as surrogates and the expected improvement method guiding optimization.

This study evaluates the effectiveness of various space-filling design strategies, particularly in high-dimensional prediction and optimization tasks. We find that uniform designs and uniform projection designs, especially those using the centered L_2-discrepancy criterion with 16 levels or more, demonstrate consistent robustness and they often outperform other types of space-filling designs. In contrast, distance-based designs like maximin distance designs underperform in higher dimensions. These findings emphasize the importance of selecting effective design strategies for high-dimensional prediction and optimization.

The paper is organized as follows. Section 2 provides background on the GP surrogate model and active learning techniques. Section 3 reviews various types of space-filling designs, followed by Section 4, which describes the test functions used. Section 5 presents experimental results, and Section 6 concludes with key findings and recommendations for future

optimization challenges.

3.2 Background

3.2.1 The surrogate model

Gaussian Processes (GPs) are well-regarded for their flexibility and accuracy in regression tasks, providing a probabilistic framework that accounts for uncertainties in predictions (Snelson 2008). The kriging model, proposed by South African geostatistician Krige (1951), is taken to be one of the surrogate models used in modelling the data. Kriging is one of the methods used to interpolate intermediate values, whereby these intermediate values are modeled using GP which is governed by prior co-variances. It provides a probabilistic prediction of the output variable, as well as an estimate of the uncertainty of the prediction (Chevalier, Picheny, and Ginsbourger 2014). The kriging predictors interpolating the observations are assumed to be noise-free (Roustant, Ginsbourger, and Deville 2012). Intermediate interpolated values obtained by kriging are the best linear unbiased predictors.

The kriging model is

$$Y(\boldsymbol{x}) = \mu(\boldsymbol{x}) + Z(\boldsymbol{x}),$$

where $\mu(\boldsymbol{x})$ is a trend function and $Z(\boldsymbol{x})$ is a stationary GP with zero mean. The ordinary kriging assumes that the trend is a constant, i.e., $\mu(\boldsymbol{x}) = \mu$, while the universal kriging assume that the trend is a linear combination of some basis functions.

The stationary GP $Z(\boldsymbol{x})$ is determined by its covariance structure $\mathrm{Cov}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma^2 r(\boldsymbol{x}_i, \boldsymbol{x}_j)$ where σ^2 is the common variance and $r(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the correlation between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. It is often assumed that $r(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a decreasing function of some distance. Various correlation functions can be used in the fitting of the kriging model. Here, we make use of the Matérn 5/2 correlation function. We fit an ordinary kriging with the constant trend. Chen et al. (2016) showed that a regression model more complex than a constant mean either has little impact on prediction accuracy or is an impediment and that the choice of correlation

function has modest effect, but there is little to separate two common choices, the power exponential and the Matérn, if the latter is optimized with respect to its smoothness.

Given n design points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and the response $\boldsymbol{y} = (y_1, \ldots, y_n)$, let $\boldsymbol{R} = (r(\boldsymbol{x}_i, \boldsymbol{x}_j))$ be the $n \times n$ correlation matrix. In this setup, for the ordinary kriging, the μ and σ^2 are estimated as follows:

$$\hat{\mu} = \frac{\mathbf{1}^\top \boldsymbol{R}^{-1} \boldsymbol{y}}{\mathbf{1}^\top \boldsymbol{R}^{-1} \mathbf{1}}, \qquad \hat{\sigma}^2 = \frac{1}{n}(\boldsymbol{y} - \hat{\mu}\mathbf{1})^\top \boldsymbol{R}^{-1}(\boldsymbol{y} - \hat{\mu}\mathbf{1}) \tag{3.1}$$

For any point $\boldsymbol{x}$, let $\boldsymbol{r}(\boldsymbol{x}) = (r(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, r(\boldsymbol{x}, \boldsymbol{x}_n))^\top$. Then the best linear unbiased predictor (BLUP) of $Y(\boldsymbol{x})$ is

$$\hat{y}(\boldsymbol{x}) = \hat{\mu} + \boldsymbol{r}(\boldsymbol{x})^\top \boldsymbol{R}^{-1}(\boldsymbol{y} - \hat{\mu}\mathbf{1}), \tag{3.2}$$

where $\hat{\mu}$ is given in (3.1). The variance of the BLUP is

$$s^2(\boldsymbol{x}) = \sigma^2 \left[1 - \boldsymbol{r}(\boldsymbol{x})^\top \boldsymbol{R}^{-1} \boldsymbol{r}(\boldsymbol{x}) + \frac{(1 - \mathbf{1}^\top \boldsymbol{R}^{-1} \boldsymbol{r}(\boldsymbol{x}))^2}{\mathbf{1}^\top \boldsymbol{R}^{-1} \mathbf{1}}\right] \tag{3.3}$$

and σ^2 can be estimated from the data as given in (3.1).

3.2.2 The expected improvement (EI)

The expected improvement is a measure of how promising a particular set of inputs is in terms of improving the objective function value. To quantify this measure, the objective function needs to be evaluated at various points, yet these evaluations are quite costly. Often when the evaluation of the objective function is costly, the need of specific strategies to optimize these functions arises. In most of these evaluations, the non-availability of derivatives prevents the use of gradient based techniques. Similarly, the use of meta-heuristics (e.g., genetic algorithm) is compromised due to severely limited evaluation budgets (Roustant, Ginsbourger, and Deville 2012). In order to curb these limitations, Jones, Schonlau, and Welch (1998) proposed the use of kriging model as the surrogate model to estimate the expected improvement of selecting a new set of inputs over the current best solution. When

carrying out minimization for example, the improvement at a new point $\boldsymbol{x}$ is

$$I(\boldsymbol{x}) = \max(y_{min} - Y(\boldsymbol{x}), 0),$$

where $y_{min} = \min(y_1, \ldots, y_n)$ is the existing minimum value. The new point will bring a positive improvement if $Y(\boldsymbol{x})$ is less than y_{min}, and an improvement of 0 otherwise. The expected improvement (EI) is simply the expectation of $I(\boldsymbol{x})$, that is,

$$\mathrm{E}[I(\boldsymbol{x})] = \mathrm{E}\left[\max(y_{min} - Y(\boldsymbol{x}), 0)\right].$$

Under the ordinary kriging, $Y(\boldsymbol{x})$ follows a normal distribution with mean $\hat{y}(\boldsymbol{x})$ and variace $s^2(\boldsymbol{x})$ given in (3.2) and (3.3), respectively. Then one can express EI in a closed form (Jones, Schonlau, and Welch 1998):

$$\mathrm{E}[I(\boldsymbol{x})] = (y_{min} - \hat{y}(\boldsymbol{x}))\Phi\left(\frac{y_{min} - \hat{y}(\boldsymbol{x})}{s(\boldsymbol{x})}\right) + s(\boldsymbol{x})\phi\left(\frac{y_{min} - \hat{y}(\boldsymbol{x})}{s(\boldsymbol{x})}\right),$$

where Φ and ϕ are the cumulative and probability density function of the standard normal distribution, respectively. The EI criterion has important properties for sequential exploration: It is null at the already visited sites, and non-negative everywhere else with a magnitude that is increasing with $s(\boldsymbol{x})$ and decreasing with $\hat{y}(\boldsymbol{x})$ (Jones, Schonlau, and Welch 1998). This guides the selection of the next set of inputs to evaluate. It encourages the exploration of regions with high uncertainty (large predicted variances) and exploitation of regions with potentially high rewards (small predicted means). The mean function and variance function of the Gaussian process are typically smooth as they are constructed based on a combination of smooth kernel functions and observed data. The cumulative distribution function $\Phi(z)$ and the probability density function $\phi(z)$ of the standard normal distribution, are also smooth functions. These functions are well-defined and infinitely differentiable for all real values of z. Therefore, combining these smooth components in the closed form EI expression results in a smooth function overall. This smoothness property of the closed form EI enables the use of gradient-based optimization methods, in particular, the L-BFGS (Limited memory Broyden-Fletcher-Goldfarb-Shanno) optimization algorithm, to search for the point of maximum expected improvement efficiently within a specified constrained domain.

3.2.3 The Efficient Global Optimization (EGO) Algorithm

This optimization method sequentially builds upon the EI criterion. EGO enhances this modeling process by intelligently selecting the most informative data points for evaluation. Rather than passively using a fixed dataset, EGO iteratively queries the objective function, focusing on areas that maximize information gain. This targeted approach reduces the number of expensive evaluations needed, accelerating the learning process. EGO (Jones, Schonlau, and Welch 1998) leverages the surrogate GP model to find the global optimum of the objective function. Using EI as the acquisition function, EGO balances the trade-off between exploring new regions of the design space and exploiting known promising areas. This results in a more efficient optimization process that converges on optimal solutions with fewer evaluations.

Algorithm 1 summarizes the procedure. Starting with an initial design $\boldsymbol{X}$ (typically, a Latin hypercube design), EGO sequentially visits a current global maximizer of EI and updates the metamodel at each iteration, including hyperparameters re-estimation (Roustant, Ginsbourger, and Deville 2012). This is done until convergence or until the budgets are exhausted.

Algorithm 1 EGO algorithm

Require: $\boldsymbol{X}$, f = function to be minimized, n_{new}=number of points to add

Evaluate f at the design points $\boldsymbol{X}$; $\boldsymbol{y} = f(\boldsymbol{X})$

Build a kriging model based on $\boldsymbol{X}$ and $\boldsymbol{y}$

for i in 1 to n_{new} **do**

Find $\boldsymbol{x}^* \leftarrow \arg\max_{\boldsymbol{x}} \mathrm{E}[I(\boldsymbol{x})]$

Evaluate $y^* \leftarrow f(\boldsymbol{x}^*)$

Update $\boldsymbol{X}$ and $\boldsymbol{y}$ with the new point $\boldsymbol{x}^*$ and response y^*

Update the kriging model

end for

Return $\boldsymbol{X}, \boldsymbol{y}$

The EGO algorithm has been shown to be effective and has been adopted in many computer experiments and are nowadays considered as reference global optimization methods in dimension $m \leq 10$ in cases where the number of objective function evaluations is drastically limited (Jones 2001).

3.3 Space-Filling Designs

We briefly review various types of space-filling designs.

Latin Hypercube Design (LHD): Based on McKay (1992)'s Latin hypercube sampling, it divides the range of each factor into bins of equal size, where n also corresponds to the number of samples to be generated resulting in a total of n^m combinations where m is the number of factors being considered. The n samples are then randomly generated such that for all one-dimensional projections, there will be only one sample in each bin. In this paper, the random Latin hypercube sampling was used to generate the random LHD with levels $0, \cdots, n-1$. In R, this was accomplished using the 'lhs' package. The following command generates an LHD with n runs and m factors.

```
> floor(lhs::randomLHS(n, m) * n)
```

Maximin Distance Designs: Introduced by Johnson, Moore, and Ylvisaker (1990), this design aims at spreading out the design points in the design space by maximizing the minimum distance between any two design points. It thus tends to place a large proportion of points at the corners and on the boundaries of the design space. Mathematically, this can be formulated as follows. Suppose we want to construct an n-run design in m factors, say $D = \{\boldsymbol{x}_1 \ldots, \boldsymbol{x}_n\}$, where $\boldsymbol{x}_i$ represnts the ith run. The maximin distance design optimizes the function below:

$$\max_{D} \min_{i<j} d\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right), \tag{3.4}$$

where $d\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)$ is the distance between the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Here we use the Euclidean distance, i.e., $d\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \sqrt{\sum_{l=1}^{m}(x_{il} - x_{jl})^2}$.

Maximin LHD: Introduced by Morris and Mitchell (1995) these designs combine the principles of Latin Hypercube Sampling (LHS) and the maximin distance criterion to optimize the spread of sample points across the design space. This hybrid approach ensures that points are uniformly distributed in each dimension, while also maximizing the minimum distance between any two points in the design. For computational purpose, Morris and Mitchell (1995) also reformatted the maximin criterion as a minimization problem given as:

$$\min_{D} \phi_p(D) = \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{d^p\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)} \right\}^{1/p} \tag{3.5}$$

where $p > 0$ is chosen large enough so that the resulting design achieves maximin distance. We use the SLHD package to generate maximin LHDs because they are better than those generated from the lhs package using the function maximinLHS. The following command generates a maximin LHD with n runs and m factors.

```
> SLHD::maximinSLHD(t = 1, n, m)$Design - 1
```

Maximum Projection Design (Maxpro Design): Although maximin LHDs ensure good space-filling in m dimensions and uniform projections in each dimension, their projection properties in two to $m-1$ dimensions are not known (Joseph, Gul, and Ba 2015). By the effect sparsity principle (Wu and Hamada 2011), only a few factors are expected to be important. To curb this, Joseph, Gul, and Ba (2015) proposed the maximum projection (MaxPro) criterion:

$$\min_{D} \psi(D) = \left\{ \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{\prod_{l=1}^{m} \left(x_{il} - x_{jl}\right)^2} \right\}^{1/m}. \tag{3.6}$$

They argued that the design that minimizes $\psi(D)$ tends to maximize its projection capability in all sub spaces of factors, and thus named these designs as maximum projection designs. We use the MaxPro package to generate a MaxPro LHD with n runs and m factors.

```
> MaxPro::MaxProLHD(n, m)$Design  * n - 0.5
```

Uniform Design (UD): These designs seek to scatter points uniformly on the domain. This is often achieved by minimizing the centered L_2-discrepancy of the design (Fang et al. 2000). For an $n \times m$ design $D = (x_{ik})$ with s levels, denoted by $0, \cdots, s-1$, its (squared) centered L_2-discrepancy is defined as

$$\begin{aligned} \mathrm{CD}(D) =& \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \prod_{k=1}^{m} \left(1 + \frac{1}{2} |z_{ik}| + \frac{1}{2} |z_{jk}| - \frac{1}{2} |z_{ik} - z_{jk}|\right) \\ & - \frac{2}{n} \sum_{i=1}^{n} \prod_{k=1}^{m} \left(1 + \frac{1}{2} |z_{ik}| - \frac{1}{2} |z_{ik}|^2\right) + \left(\frac{13}{12}\right)^m, \end{aligned} \tag{3.7}$$

where $z_{ik} = (2x_{ik} - s + 1)/(2s)$. We use the R package UniDOE to generate UDs. The following command generates a UD with n runs, m factors, and s levels.

```
> UniDOE::GenUD(n, m, s)$final_design - 1
```

The UniDOE package uses the threshold accepting algorithm to generate UDs, which often takes an excessive time in comparison with the construction of maximin and MaxPro LHDs. Thus we only used UDs for prediction, not optimization.

The UniDOE package was removed from the CRAN (Comprehensive R Archive Network) repository in 2021 as it was no longer maintained and problems were not corrected on time. Though one can still be able to install it from github.

Uniform Projection Design (UPD): Proposed by Sun, Wang, and Xu (2019), this design solely focuses on two-dimensional projections. This is due to two factor interactions being more important than three-factor or higher-order interactions. The motivating idea is that although designs with low discrepancy have good uniformity in the full-dimensional space, they can have bad projections in lower dimensional spaces, which is undesirable when only a few factors are active. Thus it is preferable to have a design with better projection properties. The uniform projection (UniPro) criterion is defined using the centered L_2-discrepancy as follows:

$$\phi(D) = \frac{2}{m(m-1)} \sum_{|u|=2} \mathrm{CD}\left(D_u\right), \tag{3.8}$$

where u is a subset of $\{1, 2, \ldots, m\}$, $|u|$ denotes the cardinality of u and D_u is the projected design of D onto dimensions indexed by the elements of u. The $\phi(D)$ is the average centered L_2-discrepancy values of all two-dimensional projections of D. The UPD scatters points uniformly in all dimensions and have good space-filling properties in terms of distance, uniformity and orthogonality (Sun, Wang, and Xu 2019). We use the UniPro package and the Differential Evolution algorithm to generate UPDs. The following command generates a UPD with n runs, m factors, and s levels.

```
> UniPro::UniPro(n, m, s)$xbest - 1
```

Note that the SLHD and MaxPro packages can only generate LHDs while the UniDOE and UniPro packages can generate balanced multi-level UDs and UPDs as long as the number of runs is a multiple of the number of levels.

3.4 Test Functions

To test the efficiency of various space-filling designs, two methods were analyzed. First the efficiency of the designs was analysed via its predictability efficiency. Here the objective was to minimize the prediction root mean square error (RMSE) to ensure accurate and reliable predictions. The second method was to look at designs through the problem of minimization convergence. Under these two methods, various test functions were taken into consideration.

3.4.1 Prediction Functions

Currin: This is a simple two dimensional polynomial function evaluated on the square $[0, 1]^2$ used for illustrating methods of modeling computer experiment output (Currin et al. 1991). It has the form:

$$f(\boldsymbol{x}) = 4.90 + 21.15x_1 - 2.17x_2 - 15.88x_1^2 - 1.38x_2^2 - 5.26x_1x_2.$$

Circuit: This is a six-dimensional function that models an output transformerless push-

pull circuit (Surjanovic and Bingham 2013) with the response being the midpoint voltage. It takes the following form:

$$f(\boldsymbol{x}) = \frac{(V_{b1} + 0.74)\,\beta\,(R_{c2} + 9)}{\beta\,(R_{c2} + 9) + R_f} + \frac{11.35R_f}{\beta\,(R_{c2} + 9) + R_f} + \frac{0.74R_f\beta\,(R_{c2} + 9)}{(\beta\,(R_{c2} + 9) + R_f)\,R_{c1}},$$
$$\text{where } V_{b1} = \frac{12R_{b2}}{R_{b1} + R_{b2}}.$$

The variables are of different domains: $R_{b1} \in [50, 150]$, $R_{b2} \in [25, 70]$, $R_f \in [0.5, 3]$, $R_{c1} \in [1.2, 2.5]$, $R_{c2} \in [0.25, 1.2]$ and $\beta \in [50, 300]$.

Piston: This is a seven-dimensional function that models the circular motion of a piston within a cylinder. It involves a chain of nonlinear functions with the response $f(\boldsymbol{x})$ being the cycle time (Surjanovic and Bingham 2013). It takes the following form:

$$f(\boldsymbol{x}) = 2\pi\sqrt{\frac{M}{k + S^2\frac{P_0V_0}{T_0}\frac{T_a}{V^2}}},$$
$$V = \frac{S}{2k}\left(\sqrt{A^2 + 4k\frac{P_0V_0}{T_0}T_a} - A\right) \text{ and } A = P_0S + 19.62M - \frac{kV_0}{S},$$

where $M \in [30, 60]$ is piston weight (kg), $S \in [0.005, 0.020]$ is piston surface area (m^2), $V_0 \in [0.002, 0.010]$ is initial gas volume (m^3), $k \in [1000, 5000]$ is spring coefficient (N/m), $P_0 \in [90000, 110000]$ is atmospheric pressure (N/m^2), $T_a \in [290, 296]$ is ambient temperature (K) and $T_0 \in [340, 360]$ is filling gas temperature (K).

Borehole: This is a 8-dimensional function that models water flow through a borehole with the response being water rate flow in m^3/yr (Surjanovic and Bingham 2013). It takes the following form:

$$f(\boldsymbol{x}) = \frac{2\pi T_u\,(H_u - H_l)}{\ln(r/r_w)\left(1 + \frac{2LT_u}{\ln(r/r_w)r_w^2K_w} + \frac{T_u}{T_l}\right)}.$$

The variable names and their corresponding domains are given as follows:

Domain	**Variable Name**
$r_w \in [0.05, 0.15]$	radius of borehole (m)
$\mathrm{r} \in [100, 50000]$	radius of influence (m)
$\mathrm{T_u} \in [63070, 115600]$	transmissivity of upper aquifer ($\mathrm{m^2/yr}$)
$\mathrm{H_u} \in [990, 1110]$	potentiometric head of upper aquifer (m)
$\mathrm{T_I} \in [63.1, 116]$	transmissivity of lower aquifer ($\mathrm{m^2/yr}$)
$\mathrm{H_I} \in [700, 820]$	potentiometric head of lower aquifer (m)
$\mathrm{L} \in [1120, 1680]$	length of borehole (m)
$\mathrm{K_w} \in [9855, 12045]$	hydraulic conductivity of borehole (m/yr)

G-function: This is an m-dimensional function designed by Ilya M. Sobol to study the sensitivity of complex mathematical models and is particularly useful for benchmarking and validating global sensitivity analysis methods. It has the form:

$$f(\boldsymbol{x}) = \prod_{i=1}^{m} \frac{|4x_i - 2| + a_i}{1 + a_i},$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ is a vector of input variables, each typically in the range $[0, 1]$ and $a_i = i/2 - 1$, for all $i = 1, \ldots, m$, are coefficients that determine the sensitivity of the function to the ith input variable. Larger values of a_i indicate less sensitivity to the corresponding input.

Wing weight: This is a function that models a light aircraft wing with the response being the wing's weight.

$$f(\boldsymbol{x}) = 0.036 S_w^{0.758} W_{fw}^{0.0035} \left(\frac{A}{\cos^2(\Lambda)}\right)^{0.6} q^{0.006} \lambda^{0.04} \left(\frac{100 t_c}{\cos(\Lambda)}\right)^{-0.3} (N_z W_{dg})^{0.49} + S_w W_p,$$

where $S_W \in [150, 200]$ is wing area ($\mathrm{ft^2}$), $W_{fw} \in [220, 300]$ is weight of fuel in the wing (lb), $A \in [6, 10]$ is aspect ratio, $\Lambda \in [-10, 10]$ is quarter-chord sweep (degrees), $q \in [16, 45]$ is dynamic pressure at cruise $\left(\mathrm{lb/ft^2}\right)$, $\lambda \in [0.5, 1]$ is taper ratio, $t_c \in [0.08, 0.18]$ is aerofoil thickness to chord ratio, $\mathrm{N_Z} \in [2.5, 6]$ is ultimate load factor, $W_{dg} \in [1700, 2500]$ is flight design gross weight (lb) and $W_p \in [0.025, 0.08]$ is paint weight $\left(\mathrm{lb/ft^2}\right)$.

Oakley & O'hagan (2004): This is a 15-dimensional function that take the following form:

$$f(\boldsymbol{x}) = \mathbf{a}_1^T \boldsymbol{x} + \mathbf{a}_2^T \sin(\boldsymbol{x}) + \mathbf{a}_3^T \cos(\boldsymbol{x}) + \boldsymbol{x}^T \mathbf{M} \boldsymbol{x},$$

where the **a**-coefficients are chosen so that 5 of the input variables contribute significantly to the output variance, 5 have a much smaller effect, and the remaining 5 have almost no effect on the output variance. Values of the coefficient vectors $\mathbf{a}_1$,$\mathbf{a}_2$ and $\mathbf{a}_3$, and the matrix $\mathbf{M}$, are available at Surjanovic and Bingham (2013).

Rosenbrock: The Rosenbrock function is a classic m-dimensional optimization problem, also known as the Valley or banana function. The global optimum lays inside a long, narrow, parabolic shaped flat valley (Molga and Smutnicki 2005). To find the valley is trivial, however convergence to the global optimum is difficult and hence this problem has been frequently used to test the performance of optimization algorithms (Picheny, Wagner, and Ginsbourger 2013). For comparison, the modified 4d version presented by Picheny, Wagner, and Ginsbourger (2013) was used. This modified version has the following form:

$$f(\mathbf{x}) = \frac{1}{3.755 \times 10^5} \left[\sum_{i=1}^{3} \left(100 \left(\bar{x}_{i+1} - \bar{x}_i^2 \right)^2 + \left(1 - \bar{x}_i \right)^2 \right) - 3.827 \times 10^5 \right], \tag{3.9}$$

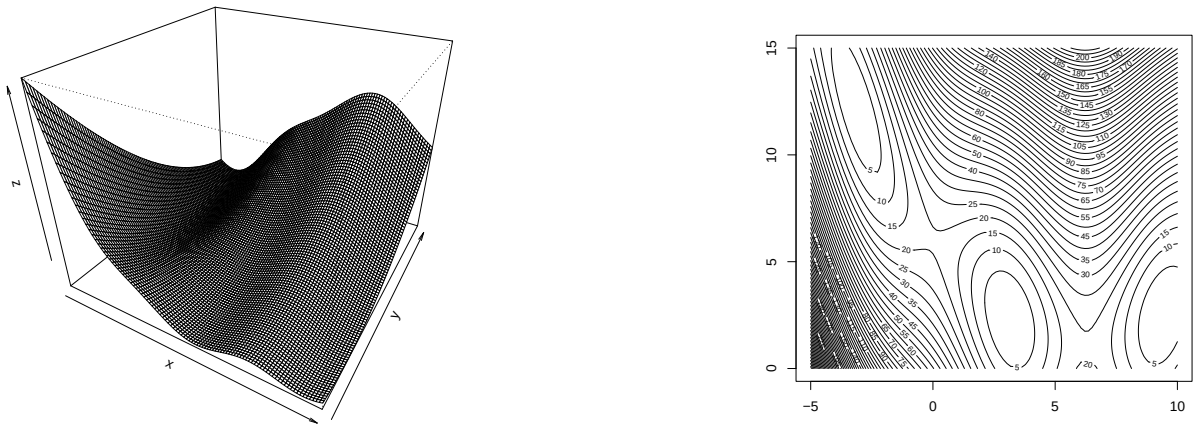
where $\bar{x}_i = 15x_i - 5$ for all $i = 1, \ldots, 4$. The test region is usually restricted to $0 \leq x_i \leq 1$ for $i = 1, \ldots, 4$.

3.4.2 Minimization Functions

Branin function: This is a well-known 2-dimensional function with multiple local minima and global minima introduced by Dixon and Szegö (1978). It is often used to test optimization algorithms due to its complexity and multi-modal nature. The Branin function has the form

$$f(\boldsymbol{x}) = f(x_1, x_2) = a \left(x_2 - b x_1^2 + c x_1 - r \right)^2 + s(1 - t) \cos\left(x_1 \right) + s$$

where the typical parameter values are $a = 1, b = 5.1/\left(4\pi^2\right), c = 5/\pi, r = 6, s = 10$ and $t = 1/(8\pi)$. The function is usually evaluated over the square $x_1 \in [-5, 10], x_2 \in [0, 15]$.



(a) 3D plot

(b) Contour plot

Figure 3.1: Branin Function

Within this domain the function has 3 global minima: $\boldsymbol{x}^* = (-\pi, 12.275), (\pi, 2.275)$ and $(9.42478, 2.475)$, with the minimum function value being $f(\boldsymbol{x}^*) = 0.397887$. The contour plot of the function is as shown in Figure 3.1.

Camel Six-Hump function: This is a 2-dimensional function with 6 local minima and 2 global minima evaluated on the rectangle $x_1 \in [-3, 3]$ and $x_2 \in [-2, 2]$. It has the form:

$$f(\boldsymbol{x}) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1 x_2 + \left(-4 + 4x_2^2\right) x_2^2.$$

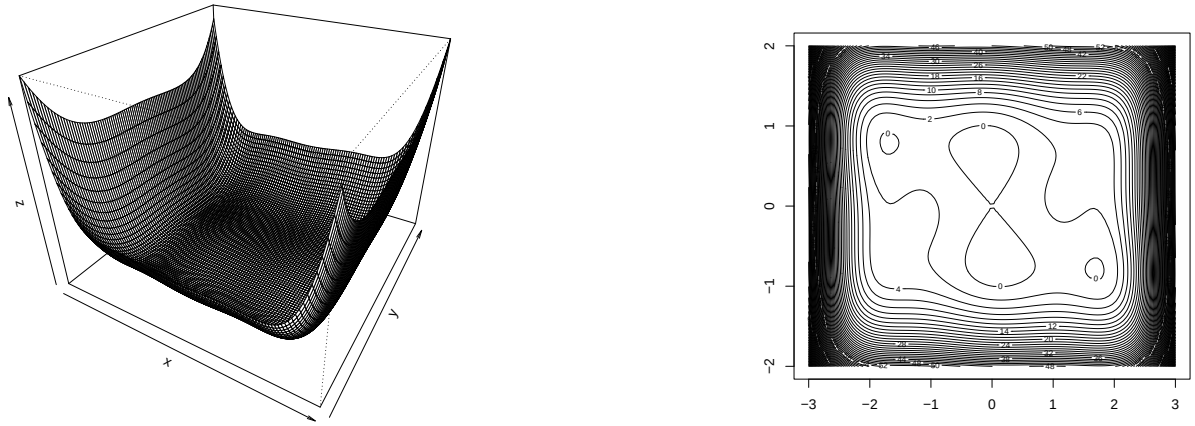
The global minima is $f(\boldsymbol{x}^*) = -1.0316$ which occurs at $\boldsymbol{x}^* = (0.0898, -0.7126)$ and $(-0.0898, 0.7126)$.

Goldstein-Price function: This is a 2-dimensional function with several local minima evaluated on the rectangle $x_i \in [-2, 2]$ for $i = 1, 2$. It has the form:

$$\begin{aligned} f(\mathbf{x}) =& \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right] \\ & \times \left[30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]. \end{aligned} \tag{3.10}$$

The global minima is $f(\boldsymbol{x}) = 3$ at $\boldsymbol{x} = (0, -1)^\top$.

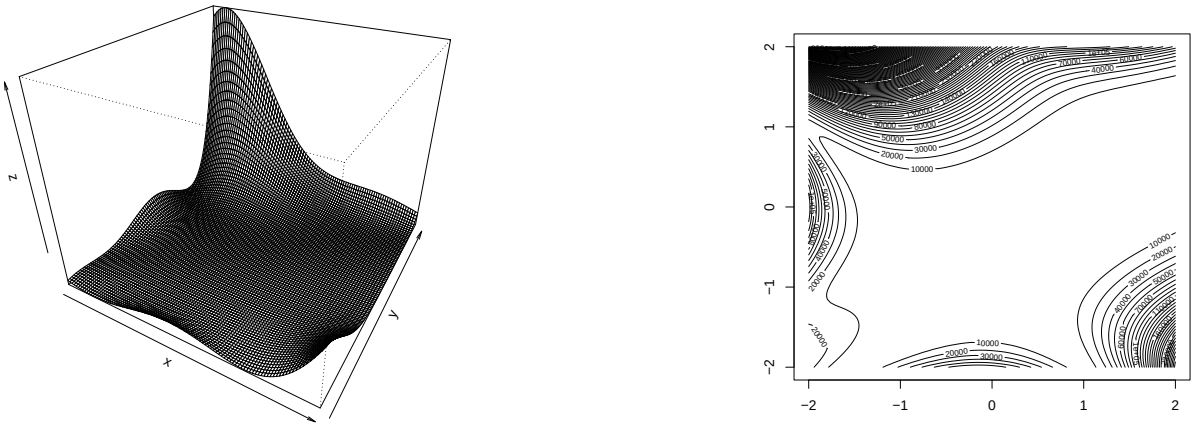
Ackley function: The Ackley function is a widely used function for testing optimization algorithms (Adorio and Diliman 2005; Molga and Smutnicki 2005). In its two-dimensional



(a) 3D plot

(b) Contour plot

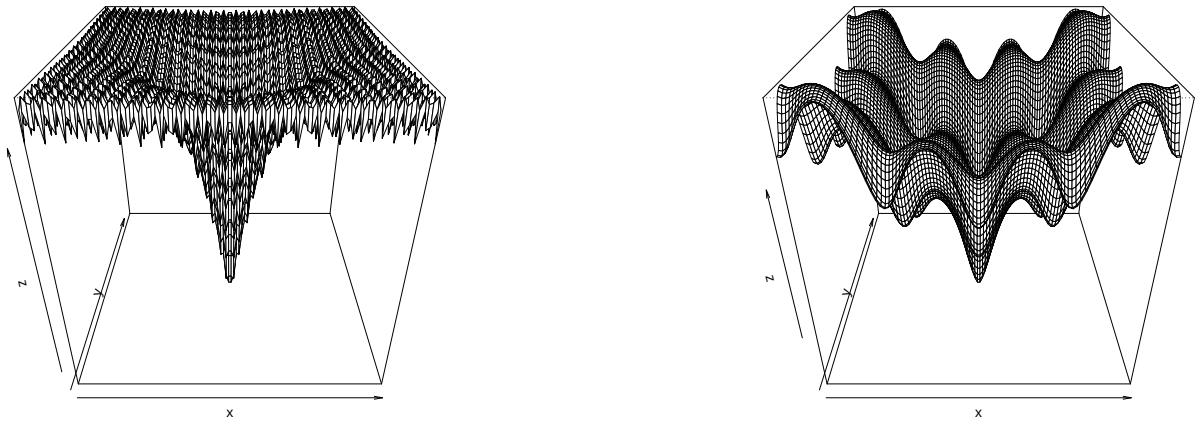
Figure 3.2: Camel Six-Hump Function



(a) 3D plot

(b) Contour plot

Figure 3.3: Goldstein-Price Function



2D Ackley function

Zoom on 2D Ackley function

Figure 3.4: Ackley function

form, it is characterized by a nearly flat outer region, and a large hole at the centre. The function poses a risk for optimization algorithms, particularly hill climbing algorithms, to be trapped in one of its many local minima (Surjanovic and Bingham 2013). The Ackley function has the following form:

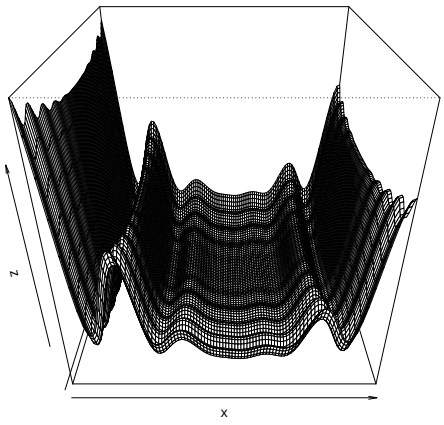
$$f(\boldsymbol{x}) = -a \exp\left(-b\sqrt{\frac{1}{m}\sum_{i=1}^{m} x_i^2}\right) - \exp\left(\frac{1}{m}\sum_{i=1}^{m}\cos\left(cx_i\right)\right) + a + \exp(1)$$

with the recommended values for a, b and c being $a = 20$, $b = 0.2$ and $c = 2\pi$. It is usually evaluated on the hypercube $x_i \in [-32.768, 32.768]$ for all $i = 1, \cdots, m$ with a global minimum of $f(\boldsymbol{x}) = 0$ at $x_i = 0$ for $i = 1, \cdots, m$.

Levy function: This is an m-dimensional function often used as a test function for optimization. It is usually evaluated on the hypercube $x_i \in [-10, 10]$ $\forall i = 1, \ldots, m$ and has the global minimum of $f(\boldsymbol{x}^*) = 0$ at $\boldsymbol{x}^* = (1, \ldots, 1)^\top$. It is defined as

$$f(\boldsymbol{x}) = \sin^2(\pi w_1) + \sum_{i=1}^{m-1}(w_i - 1)^2\left[1 + 10\sin^2(\pi w_i + 1)\right] + (w_m - 1)^2\left[1 + \sin^2(2\pi w_m)\right],$$

where $w_i = 1 + (x_i - 1)/4$ for $i = 1, \ldots, m$.



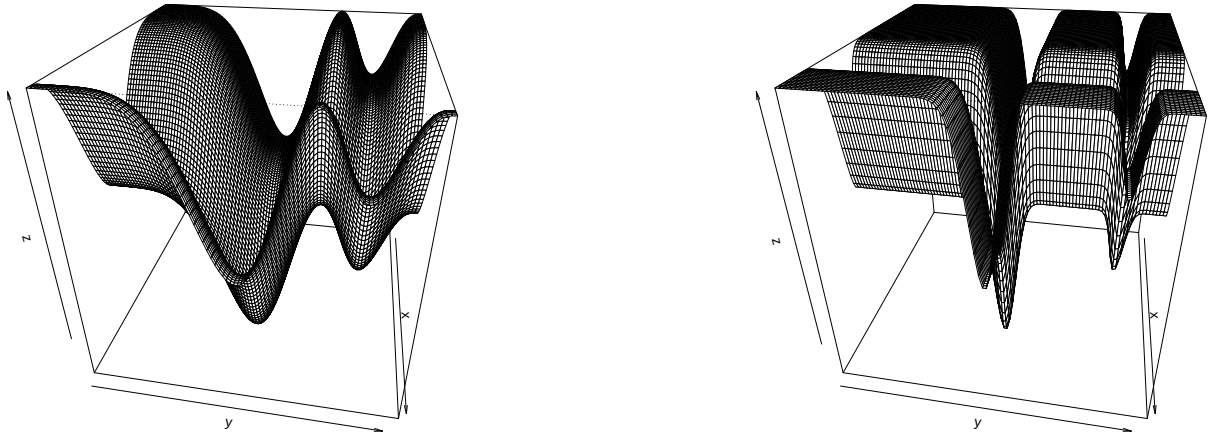
(a) 2D levy Function

Figure 3.5: Levy function

Michalewicz Function: This is a multimodal test function with $m!$ local optima for m dimensions. It is characterized by its steep valleys and ridges. The "steepness" of the valley or edges is defined by the parameter p (Molga and Smutnicki 2005). For larger p it is quite difficult to obtain the optimum as the function values for points in the space outside the narrow peaks give very little to no information on the location of the global optimum. At the same time an increase in dimensionality increases the difficulty due to the increased number of local minima. The function is defined as:

$$f(\boldsymbol{x}) = -\sum_{i=1}^{m} \sin(x_i) \left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{2p}$$

with p usually set at $p = 10$ and the domain restricted to $x_i \in [0, \pi]$ for $i = 1, \cdots, m$. The global minimum is approximated to be $f(\boldsymbol{x}) = -1.8013$ for $m = 2$, $f(\boldsymbol{x}) = -4.6877$ for $m = 5$ and $f(\boldsymbol{x}) = -9.6602$ for $m = 10$.



(a) 2D Michalewicz function for $p = 1$

(b) 2D Michalewicz function for $p = 10$

Figure 3.6: Michalewicz function

3.5 Numerical Experiments

This section presents the numerical results from applying Gaussian Process (GP) to the prediction functions and Efficient Global Optimization (EGO) to the minimization functions. In order to determine the effectiveness of the different initial designs, both prediction and optimization functions were taken into consideration.

We first generate different types of space-filling designs using various packages as described in Section 3.3 and examine their performance and relationship under different criteria. For each type of design, we ran the corresponding algorithm to construct 80×8 LHDs for 100 times. Table 3.1 shows the mean and median of the various design criteria for each design type. The design criteria are the maximin criterion (3.5) with $p = 15$, the MaxPro criteron (3.6), the centered L_2-discrepancy (3.7), the UniPro criterion (3.8), and the absolute average correlation criterion (3.11) defined below:

$$\rho_{ave} = \frac{2}{m(m-1)} \sum_{k=1}^{m} \sum_{l=k+1}^{m} \frac{\left| \sum_{i=1}^{n} \left(x_{ik} - \bar{x}_{\cdot k}\right)\left(x_{il} - \bar{x}_{\cdot l}\right) \right|}{\sqrt{\sum_{i=1}^{n} \left(x_{ik} - \bar{x}_{\cdot k}\right)^2 \sum_{i=1}^{n} \left(x_{il} - \bar{x}_{\cdot l}\right)^2}}. \tag{3.11}$$

For all criteria, smaller values represent better designs.

As expected, each type of design performs the best under the corresponding criterion used to generate the design. For example, maximin designs have smaller $\phi_p(D)$ values and MaxPro designs have smaller $\psi(D)$ values. In addition, random LHDs are worse than all other types of space-filling designs. Regarding to the ρ_{ave} criterion, UPDs and uniform designs (ud) are considerably better than other types of designs. Figure 3.7 gives a visualization where UPDs are fairly comparable to the uniform designs. These two types of designs have the highest correlation. From Table 3.1 and Figure 3.7, we are capable to determine that uniform designs and UPDs are more robust than other types of designs under various design criteria. We have examined cases with different design sizes, and the conclusions are similar.

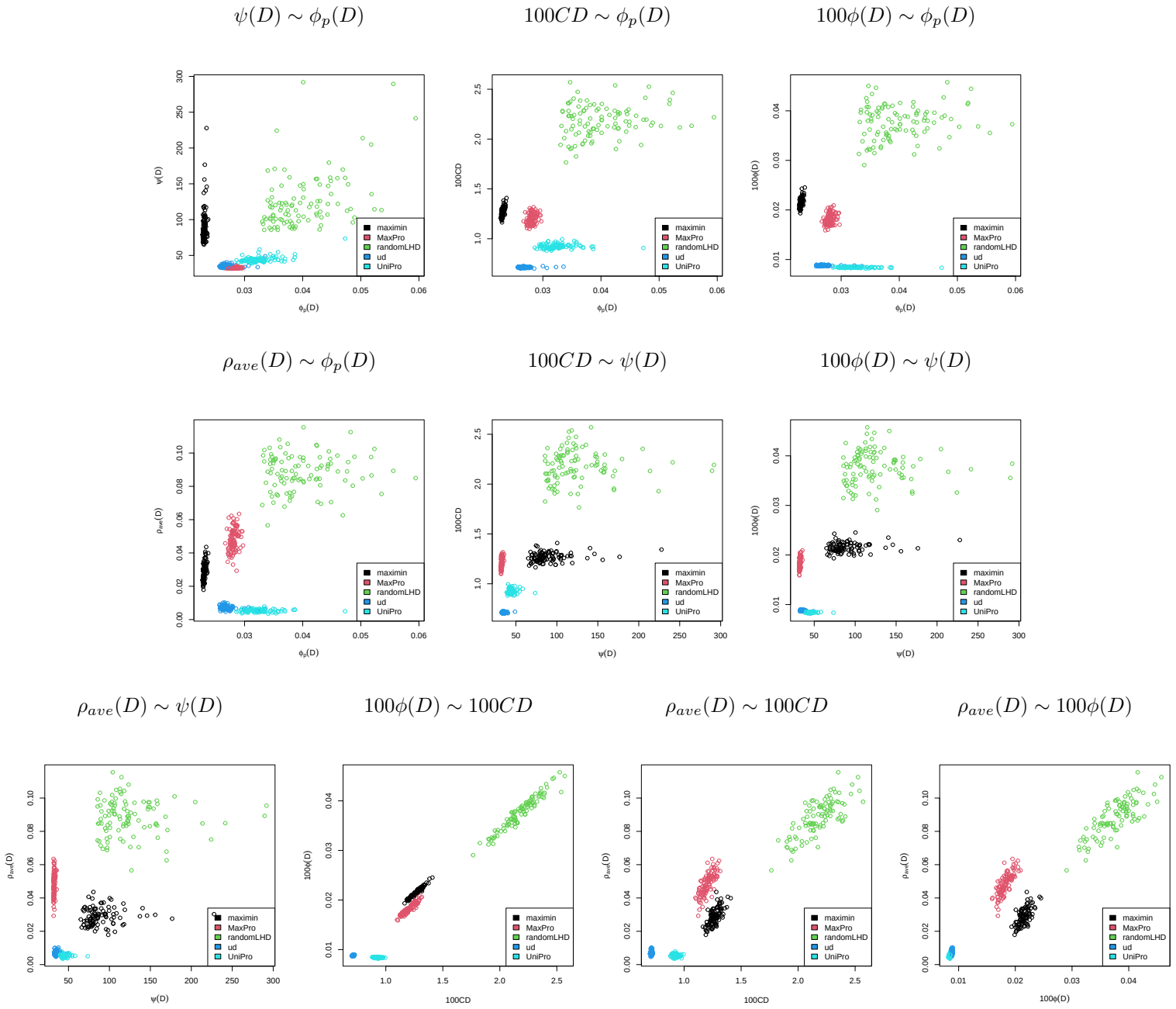


Figure 3.7: Comparison of 100 80×8 LHDs using various criteria

Table 3.1: Comparison of 100 80×8 designs using various criteria

design	**summary**	$\phi_p(D)$	$\psi(D)$	$100CD$	$100\phi(D)$	ρ_{ave}
maximin	mean	0.0231	90.5940	1.2676	0.0215	0.0289
	median	0.0231	83.5325	1.2616	0.0214	0.0289
MaxPro	mean	0.0281	32.5730	1.2028	0.0182	0.0483
	median	0.0280	32.3859	1.1962	0.0181	0.0481
ud	mean	0.0269	35.0794	0.7129	0.0088	0.0072
	median	0.0266	34.6913	0.7131	0.0088	0.0071
UniPro	mean	0.0326	44.4252	0.9240	0.0084	0.0054
	median	0.0321	43.6042	0.9215	0.0084	0.0054
randomLHD	mean	0.0402	125.7954	2.1808	0.0377	0.0883
	median	0.0385	116.8441	2.1881	0.0379	0.0891

3.5.1 Prediction results

The test functions are evaluated on various space-filling LHDs of sizes 64×15 and 128×31. In addition, UPDs with 8 and 16 levels are also constructed and used for model fitting and prediction with the purpose of evaluating whether the number of levels affects the performace. All designs are scaled to the function domain accordingly. Since the functions are of various dimensions, yet the designs used are of a pre-specified dimension, the unused dimensions are considered to be inert and serve as 'noise'. This is quite common in practice whereby some inert factors are involved in an experimental study. For example, the borehole function has 8 variables, so we have 7 inert variables when the design has 15 columns. For each design type, an ordinary kriging model with a constant trend using the matérn correlation function with $\nu = 5/2$ is then fitted. To evaluate the performance of different types of designs, the normalized root mean square error is used, which is given by

$$\text{Normalized RMSE} = \left[\frac{N^{-1} \sum_{i=1}^{N} \left\{ \hat{y}\left(\boldsymbol{x}_i\right) - y\left(\boldsymbol{x}_i\right) \right\}^2}{N^{-1} \sum_{i=1}^{N} \left\{ \bar{y} - y\left(\boldsymbol{x}_i\right) \right\}^2} \right]^{1/2},$$

where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are the inputs of the test dataset, $y(\boldsymbol{x}_i)$ is the true response, $\hat{y}(\boldsymbol{x}_i)$ is the predicted value from the GP model, and $\bar{y}$ is the mean of the responses in the training dataset. This criterion is related to R^2 in regression, but it measures performance for a new test dataset and smaller values are desirable (Chen et al. 2016). We used a random LHD with $N = 10,000$ runs as the test dataset.

In order to reduce the randomness effect, the process is replicated 100 times. The replication of the design generation is done via row and column permutations. For each design type, except for the random LHD, 100 designs are generated and the best design among these 100 generated designs determined by each design type criterion is chosen as the candidate design. Using row and column permutation of the best design for each design type, 100 designs are thus generated. This method of design generation is chosen as it is almost infeasible to generate uniform designs of size 64×15 and 128×31 for each replication. In particular to generate a 10×3 uniform design with 10 levels using a machine equipped with

a 13th Gen Intel Core™ i7-1360P 16-core CPU running at 2.2 GHz it takes an average of 5.8 seconds while a machine with Intel® Xeon® Platinum 8160 CPU with 48 cores running at 2.10 GHz takes an average of 5.77 seconds. In order to use the row and column permutation, a fairly large enough design is to be used. For example to generate a 80×8 UD design with 80 levels requires ≈ 439 minutes (7.32 hours) just for one design in the first machine, while needing 432 minute (7.2 hours) in the second machine. This shows as to why permutation is the best way to do the design generation.

Figure 3.8 shows the normalized RMSEs for the various test functions using various 64×15 designs. The mean and median RMSEs are also obtained and are reported in Table 3.2. A general conclusion is that the use of uniform designs and uniform projection designs in prediction is efficient as they result in minimal normalized RMSEs for most of the test functions, compared to the other types of designs. This could be attributed to their robustness in nature. On the other hand, the random LHD is often the worst as expected with one exception. For the 9-dimensional G-function, the maximin, MaxPro and uniform LHDs perform worse than the random LHD whereas all three of the UPDs perform better than the random LHD. In particular the 8-level UPD (upd8q) performs the best in this case.

Among all LHDs, the UPD appears to be most robust as it is never worse than the random LHD. Also it is better than the maxmin and MaxPro designs with one exception for the Wingweight function, where the maximin design is better. In addition, the 16-level UPD (upd16q) performs better than the 8-level UPD except for the case of the G-function. At the same time 16-level UPD performs better than the 64-level UPD. This gives the notion of the importance of the number of levels. A fair enough number of levels in comparison to the number of runs is important in prediction. While in physical experiments, 3-5 levels are enough, in computer experiments, more levels are needed.

The conclusions reached are further cemented by observing the results obtained when using 128×31 design size for training and predicting on the test dataset with 10,000 runs. Here 8-level UPD is not used as it produces many outliers which would distort the visual

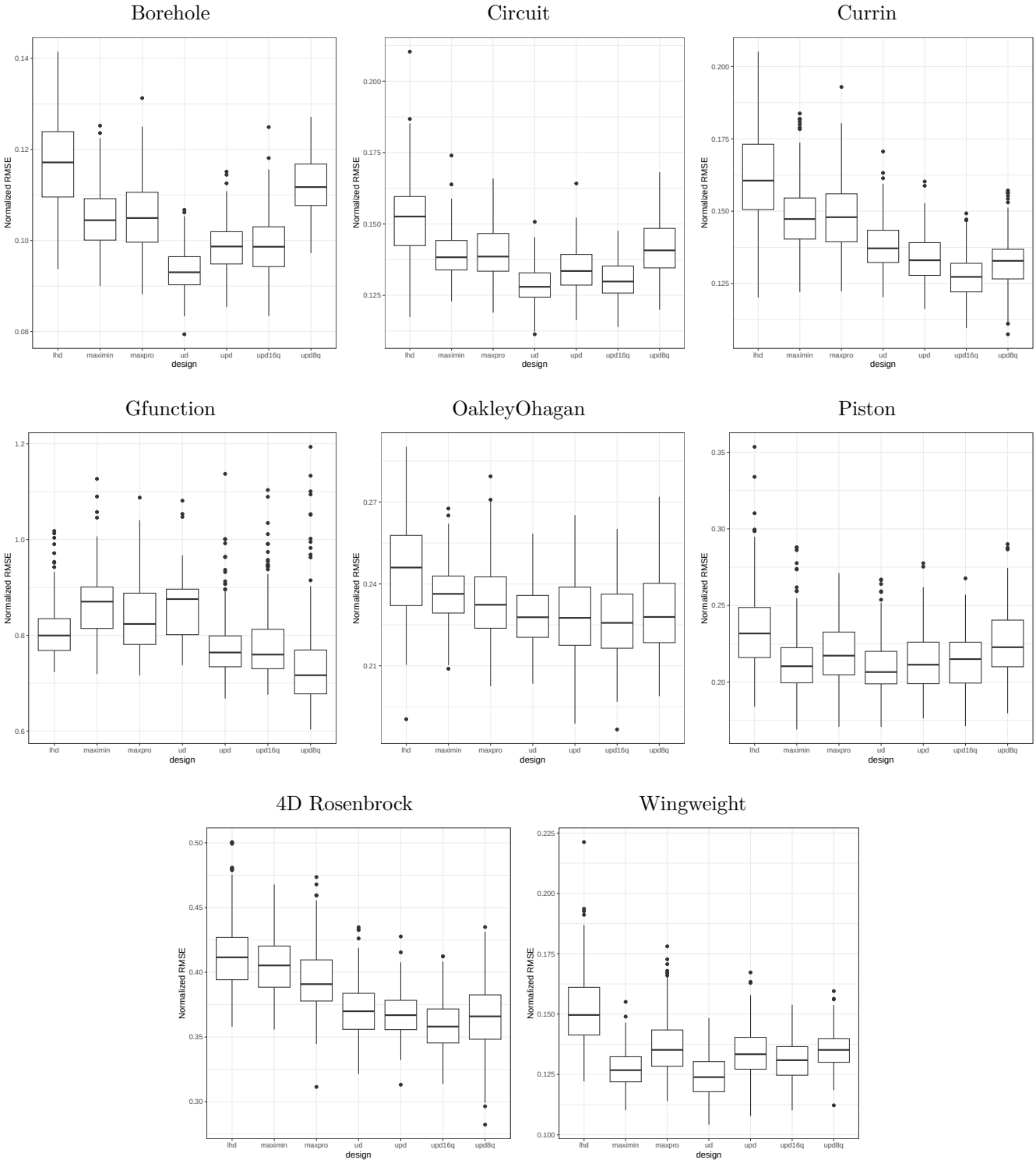


Figure 3.8: Normalized RMSEs for various test functions and 64×15 designs

Table 3.2: Means and medians of normalized RMSEs for 64×15 designs

fun		lhd	maximin	maxpro	ud	upd	upd16q	upd8q
Borehole	mean	0.1168	0.1048	0.1053	**0.0936**	0.0986	0.0990	0.1119
	median	0.1171	0.1044	0.1049	**0.0930**	0.0987	0.0986	0.1117
Circuit	mean	0.1519	0.1394	0.1397	**0.1286**	0.1337	0.1308	0.1417
	median	0.1526	0.1383	0.1386	**0.1280**	0.1335	0.1298	0.1408
Currin	mean	0.1622	0.1487	0.1480	0.1380	0.1333	**0.1273**	0.1326
	median	0.1606	0.1473	0.1479	0.1372	0.1330	**0.1273**	0.1328
Gfunction	mean	0.8092	0.8618	0.8328	0.8540	0.7795	0.7830	**0.7423**
	median	0.7996	0.8704	0.8237	0.8757	0.7642	0.7598	**0.7167**
OakleyOhagan	mean	0.2461	0.2358	0.2329	0.2284	0.2277	**0.2258**	0.2300
	median	0.2460	0.2364	0.2324	0.2278	0.2276	**0.2257**	0.2279
Piston	mean	0.2351	0.2127	0.2195	**0.2101**	0.2137	0.2146	0.2254
	median	0.2317	0.2103	0.2172	**0.2065**	0.2113	0.2149	0.2227
Rosenbrock	mean	0.4134	0.4051	0.3943	0.3708	0.3672	**0.3583**	0.3655
	median	0.4115	0.4053	0.3908	0.3699	0.3669	**0.3580**	0.3659
Wingweight	mean	0.1520	0.1275	0.1367	**0.1244**	0.1337	0.1310	0.1355
	median	0.1496	0.1268	0.1352	**0.1239**	0.1334	0.1309	0.1352

comparison of the results. The 16-level UPD still has the best performance in most of the test functions; see Table 3.3 and Figure 3.9.

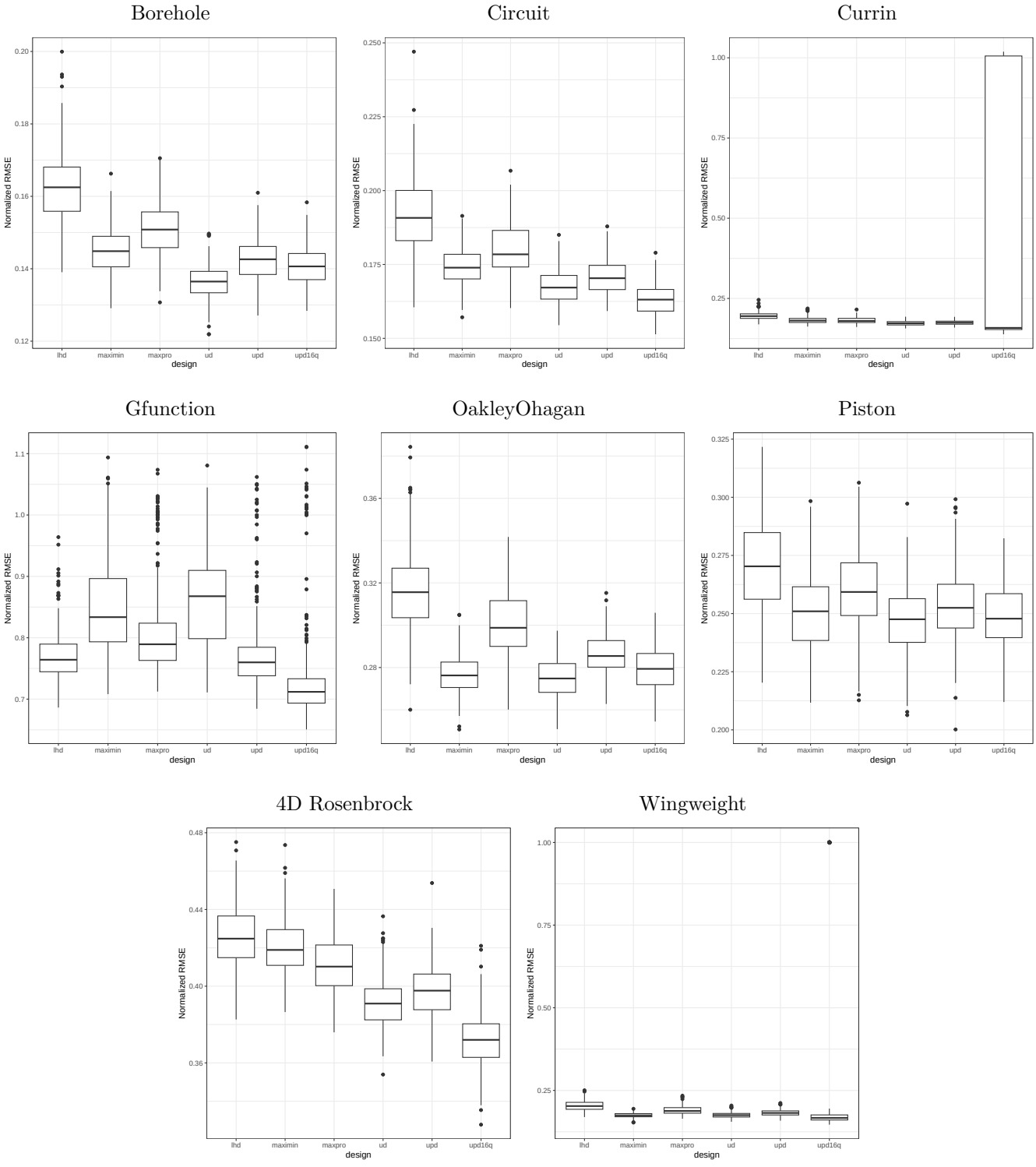


Figure 3.9: Normalized RMSEs for various test functions and 128×31 designs

Table 3.3: Means and medians of normalized RMSEs for 128×31 designs

fun		lhd	maximin	maxpro	ud	upd	upd16q
Borehole	mean	0.1624	0.1451	0.1511	**0.1366**	0.1425	0.1408
	median	0.1625	0.1449	0.1508	**0.1365**	0.1426	0.1406
Circuit	mean	0.1920	0.1742	0.1798	0.1676	0.1709	**0.1630**
	median	0.1908	0.1739	0.1784	0.1672	0.1704	**0.1631**
Currin	mean	0.1960	0.1818	0.1815	**0.1730**	0.1747	0.4376
	median	0.1949	0.1809	0.1799	0.1724	0.1744	**0.1582**
Gfunction	mean	0.7697	0.8550	0.8074	0.8682	0.7724	**0.7297**
	median	0.7642	0.8336	0.7894	0.8677	0.7599	**0.7119**
OakleyOhagan	mean	0.3166	0.2769	0.3006	**0.2752**	0.2865	0.2797
	median	0.3156	0.2763	0.2988	**0.2748**	0.2854	0.2794
Piston	mean	0.2699	0.2504	0.2601	**0.2475**	0.2535	0.2483
	median	0.2703	0.2510	0.2593	**0.2475**	0.2525	0.2478
Rosenbrock	mean	0.4257	0.4204	0.4105	0.3916	0.3975	**0.3724**
	median	0.4248	0.4189	0.4102	0.3909	0.3977	**0.3720**
Wingweight	mean	0.2037	**0.1748**	0.1900	0.1756	0.1821	0.3112
	median	0.2027	0.1745	0.1879	0.1752	0.1819	**0.1669**

From Figure 3.9 we see that the Currin and the wing weight functions produce a few large RMSE values, likely due to the failure of the optimization routine when fitting the

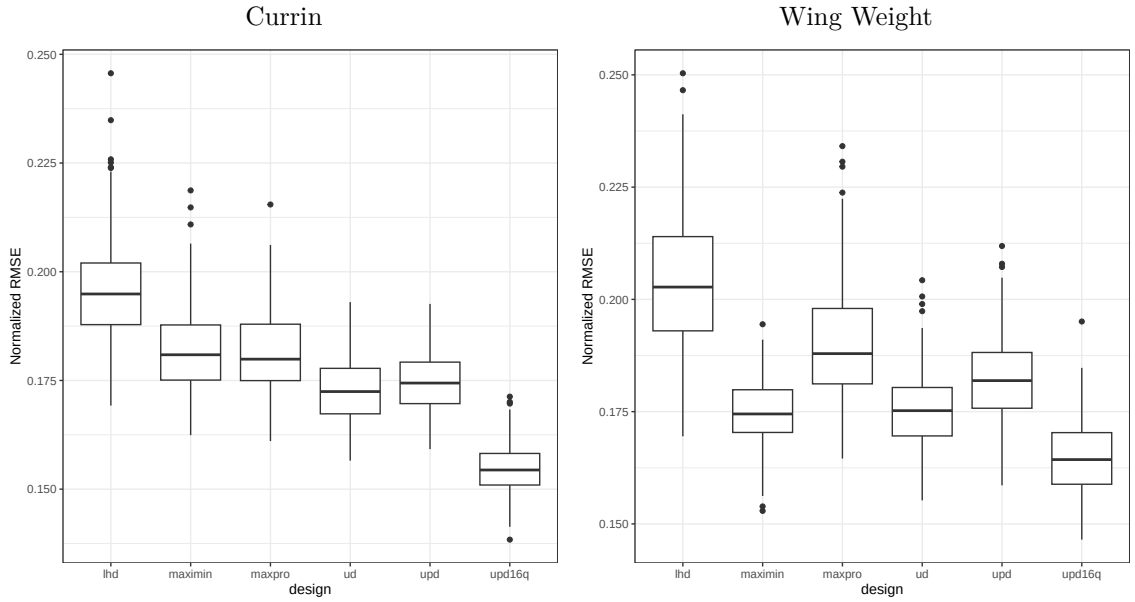


Figure 3.10: Normalized RMSEs for Currin and Wing Weight functions without the outliers

kriging model. These outliers distort the display of the boxplots even though the other RMSE values are generally concentrated below the RMSE value of the other functions. For a fair comparison, we got rid of these outliers and plotted the rest as shown in Figure 3.10.

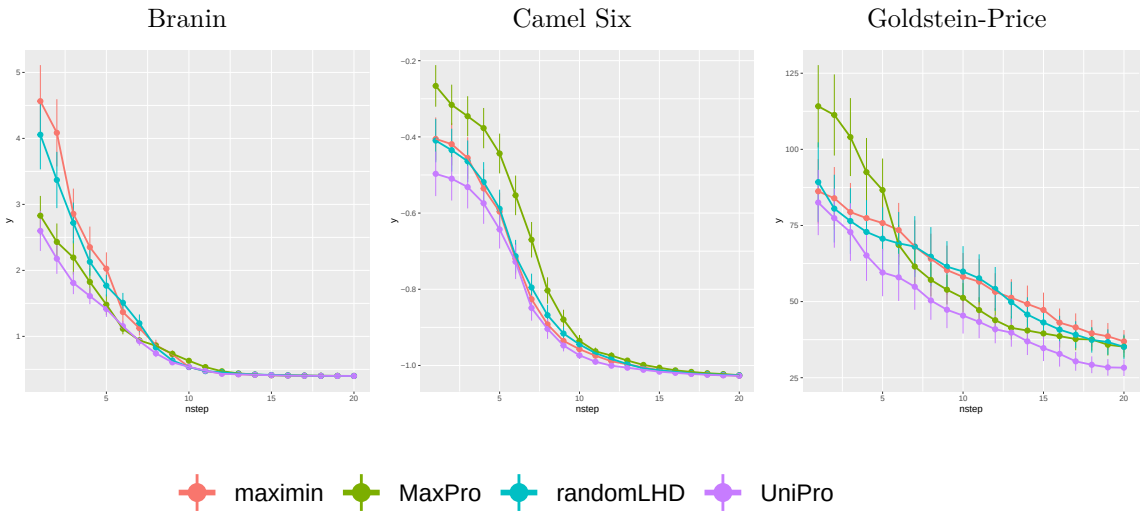


Figure 3.11: Minimization path for 2 dimensional test functions

3.5.2 Minimization results

We consider four types of space-filling designs for sequential minimization: random LHDs, maximin LHDs, MaxPro LHDs and UniPro LHDs. The test functions are evaluated at $n_0 = 10 \times m$ initial points, apart from the Branin function which is evaluated at $n_0 = 10$ initial points. These functions are then minimized using the EGO approach. We use the function fastEGO.nsteps in the R DiceOptim package to sequentially add a point at a time with n_1 steps. For each step, the cumulative minimum is recorded. This process is replicated 200 times.

Figure 3.11 shows the average minimal values at each step, together with the 2 standard error bar, of the resulting minimization path with $n_1 = 20$ added points for three 2-dimensional test functions. For the Branin function, MaxPro and UniPro designs yield smaller y-values at early stages than maximin and random LHDs. The difference gradually diminishes over the sequential optimization process. After 15 additional points, all the methods have the same y-value indicating that the function minimum has been achieved. The result is similar for the Camel Six function. For the Goldstein-Price function, the UniPro

design maintains its advantage over others after 20 steps.

The results presented above are not unique. They are determined by the computation of the first y-value from the initial design. The design that initially produces the least function value tends to generally converge faster to the function minimum, as the sequential points majorly depend on the expected improvement with influence from the design type. As the number of iterations (nsteps) increases, the influence of the initial design type on the optimization process diminishes. Initially, the choice of design strategy can play a significant role in guiding the search for the global optimum. However, as more sample points are iteratively added based on the acquisition function, the optimization process becomes increasingly driven by the updated GP model. This model incorporates all collected data points, thereby reducing the relative impact of the initial design. Consequently, the optimization converges towards the global optimum, and differences attributed to the initial design strategy become less pronounced.

Though this is the case, in high dimensional setup, or using complicated functions, the number of sequentially added points needed for convergence might be quite large. With a limited budget, the effect of initial design could be profound.

Figure 3.12 shows the minimization path for 3 other test functions with varying dimensions from 4 to 8. One striking observation is the decreasing efficiency of using maximin designs when the dimension increases. At lower dimensions, maximin designs fairly compete with the other designs but as the dimensions increase to 6 and 8, it is observed that the maximin designs do poorly. This is even after increasing the number of steps to 30 or more. We also note that MaxPro designs are worse than random LHDs and UniPro designs for the Ackley and Levy function. Overall, UniPro designs are robust and perform well under all situations, especially in high dimensions and complicated test functions.

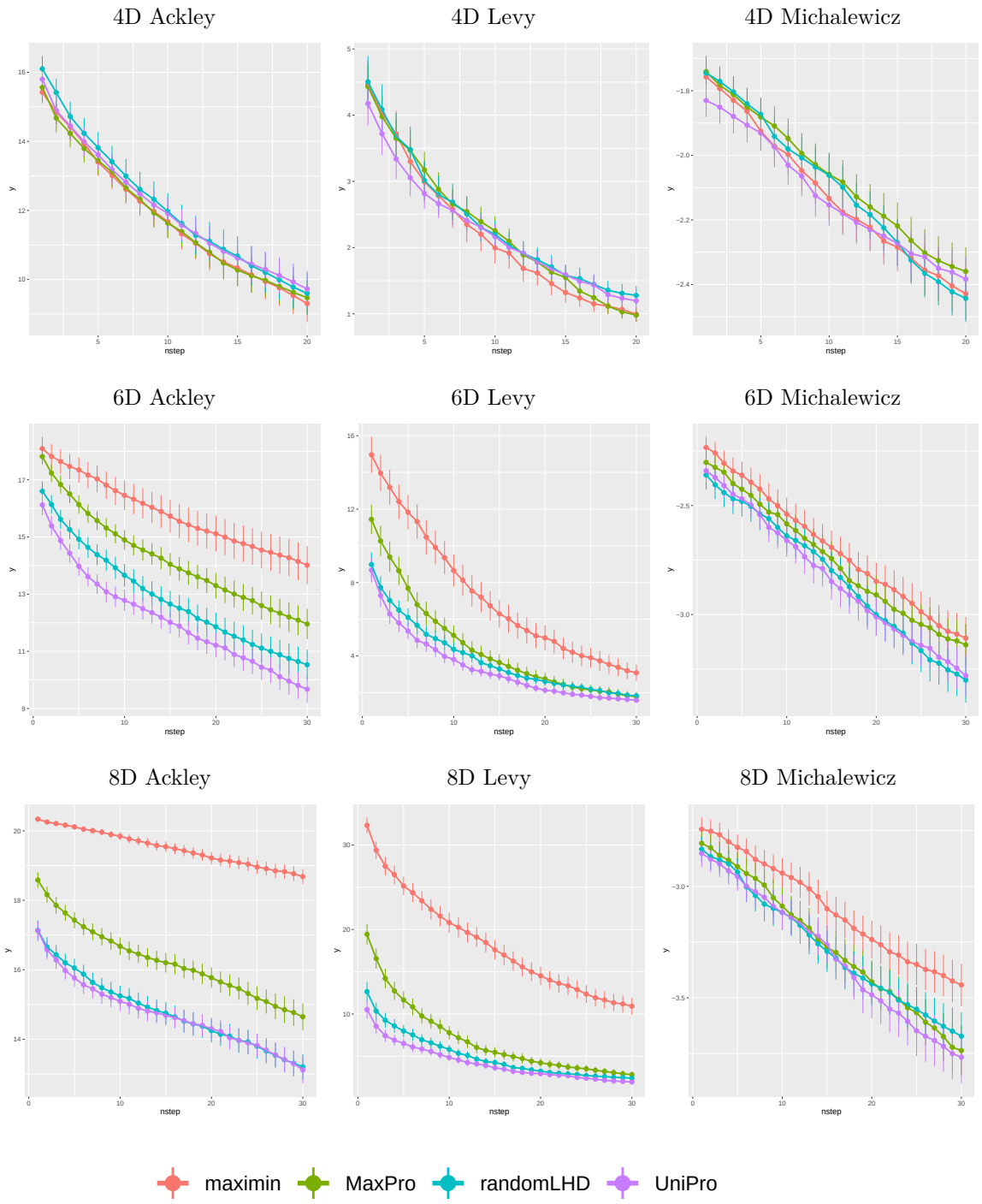


Figure 3.12: Minimization path for test functions with varying dimensions

3.6 Concluding Remarks

In conclusion, the numerical evaluation of Gaussian Process (GP) and Efficient Global Optimization (EGO) techniques reveal that the uniform design (UD) and uniform projection design (UPD) stand out for their robustness and efficiency. When compared across various design criteria, the UD and UPD consistently demonstrate superior performance. The extensive experimentation, including the generation of 100 designs and their permutations, highlight the efficiency of the uniform designs in reducing normalized root mean square error in prediction tasks. Notably, the UPD with 16 levels proved to be particularly effective, underscoring the importance of having a sufficiently large number of levels in design, yet it is not necessary to use LHDs with the number of levels being equal to the number of runs when the latter is large.

The comparative analysis of different initial designs for EGO further confirmed the efficacy of UPDs. As evidenced by the results, UPDs converged more quickly to the function minimum compared to other design methods. Although the differences in optimization outcomes among design strategies became less pronounced with more iterations, the initial design choice had a significant impact at early stages, which could prolong to later stages when we encounter complicated optimization tasks in high dimensions. This observation underscores the value of selecting a good design strategy for enhancing the efficiency of the optimization process. In conclusion, UPDs are simple to construct, offering a feasible solution for large-scale, high-dimensional prediction and optimization tasks.

However, the results also revealed some limitations, particularly with distance-based designs like the maximin criterion in high-dimensional spaces. One reason as to why this is the case is because of the use of Euclidean distance as the criterion to be optimized by the maximin design yet it is well documented that Euclidean distance is not a good metric in high dimensions. In high dimensions, the natural intuition which comes from our three-dimensional world fails and thus do not apply (Domingos 2012). Taking an example of Gaussian distribution, in high dimensions, most of the mass of a multivariate Gaussian

distribution is not near the mean, but in an increasingly distant "shell" around it; and most of the volume of a high dimensional orange is in the skin, not the pulp. If a constant number of points is distributed uniformly in a high-dimensional hypercube, beyond some dimensionality most points are closer to a face of the hypercube than to their nearest neighbor. When a hypersphere is approximated by inscribing it in a hypercube, in high dimensions almost all the volume of the hypercube is outside the hypersphere (Domingos 2012). At the same time, in high dimensions, the ratio between the nearest and farthest points using Euclidean distance metric approaches 1, i.e., the points essentially become uniformly distant from each other (Aggarwal, Hinneburg, and Keim 2001). Thus, as all points are essentially uniformly distant from each other, the distinction is meaningless. This indicates as to why the Euclidean distance would therefore not be a good measure of distance in high dimensions. Since the spreading of points using the maximin design is done using the Euclidean distance, these points will mostly cover the empty space rather than the surface where the function is defined, hence making it be inefficient as compared to the other methods. Perhaps this phenomenon also impacts the MaxPro designs, making the efficiency of the MaxPro design to decline as the dimensions increases as seen in Figure 3.12. Though this is not known.

As the UPD focusing on 2-dimensional uniformity, it is not highly impacted by the curse of dimensionality. Thereby retaining its high performance in comparison to the distance based designs with regards to minimizing high dimensional functions. The inefficacy of the maximin design in higher dimensions, due to the limitations of Euclidean distance metrics, suggests that such distance-based methods may not be well-suited for complex high-dimensional problems. Conversely, the UPD's use of the centered L_2-discrepancy criterion allows it to mitigate the curse of dimensionality, maintaining its effectiveness across various dimensions. This highlights the need for careful consideration of design criteria and metrics in high-dimensional optimization tasks to ensure accurate and efficient results.

CHAPTER 4

Kriging Based Sequential Region Shrinkage with EGO for Hyperparameter Optimization

4.1 Introduction

Black-box optimization is a key challenge in many fields, including engineering, finance, and machine learning. It involves finding the optimal solution to a complex function that is either expensive or impossible to evaluate analytically. Formally, this can be expressed as:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x}),$$

where Ω represents the search space of $\boldsymbol{x}$. Optimizing $f(\boldsymbol{x})$ is a non-trivial task, typically requiring either derivative-based or derivative-free methods. Derivative-based methods depend on the calculation of the objective function's derivatives, making them suitable when $f(\boldsymbol{x})$ is smooth and differentiable, and its derivatives are easy to compute. However, in many real-world scenarios, the function is unknown or difficult to differentiate, making derivative-free methods preferable. These methods are especially useful in cases of black-box functions, where the objective is not directly accessible.

Hyper-parameter Optimization (HPO) is a classic example of a black-box optimization problem, where the goal is to select the best set of hyper-parameters for a learning algorithm. These hyper-parameters are typically set before training and control various aspects of the model, such as learning rate, regularization, and the choice of the optimization algorithm. The model's performance can vary significantly based on the hyper-parameters chosen (Feurer and Hutter 2019), with results sometimes fluctuating drastically depending on the

architecture (Liu, Simonyan, and Yang 2018). Several methods are widely used for HPO, including grid search, random search (Bergstra and Bengio 2012), and Bayesian optimization (Pelikan, Goldberg, Cantú-Paz, et al. 1999).

Grid search systematically evaluates the model's performance for every combination of hyper-parameters within a predefined grid, choosing the configuration that yields the best result. Random search, on the other hand, samples hyper-parameters randomly from a predefined distribution and selects the best-performing configuration. Bayesian optimization constructs a probabilistic model of the objective function by combining prior knowledge with previously evaluated configurations. The posterior model is then optimized using an acquisition function, and the process is repeated until no further improvements can be made (Brochu, Cora, and De Freitas 2010).

While grid search and random search are simple and easy to implement, they become computationally expensive, especially in high-dimensional hyper-parameter spaces. Bayesian optimization is typically more efficient and effective for HPO (Snoek, Larochelle, and Adams 2012). One way to enhance the efficiency of Bayesian optimization is through the use of a surrogate model, which approximates the objective function and directs the search toward promising regions of the hyper-parameter space (Jones, Schonlau, and Welch 1998).

Bayesian optimization has gained popularity in solving black-box optimization problems due to its ability to handle noisy and non-convex functions, which are common in real-world scenarios. The probabilistic surrogate model used in Bayesian optimization is typically a Gaussian Process (GP), which is a flexible and powerful tool for modeling complex functions (Rasmussen and Williams 2006). GPs can predict the value of the objective function at unexplored points while also providing an estimate of the uncertainty of these predictions.

In this paper, we propose a Kriging-based region shrinkage method that builds on Efficient Global Optimization (EGO) (Jones, Schonlau, and Welch 1998). The method sequentially refines the region of interest (ROI) based on a proportion of most informative data points, progressively shrinking the search space by reducing the size of the interval for each

hyper-parameter at each step. This approach allows us to focus more precisely on promising regions. By using EGO, the method balances exploration and exploitation within the domain.

The effectiveness of the proposed method is demonstrated using several well-known physical test functions from the Virtual Library of Simulation Experiments (Surjanovic and Bingham 2013). We compare our approach to existing optimization methods and show that the proposed derivative-free method achieves results on par with or exceeding expectations. Empirical results on DE hyperparameter optimization for constructing uniform projection designs show that our method requires fewer computational resources while performing comparably to, or better than, traditional hyper-parameter tuning methods such as grid search and random search. Although the theoretical guarantees are limited, the empirical results indicate strong potential for practical applications.

4.2 Background Theories

4.2.1 Related Work

In pursuit of minimizing loss, numerous optimization techniques based on the Bayesian approach have been developed. While some methods emphasize dimensionality reduction to enhance computational efficiency, the majority focus on reducing the size of the search space, commonly referred to as variable interval size reduction. The work presented here falls within this category.

Among the various strategies explored are the Controlled Gutmann-RBF (CG-RBF) method (Regis and Shoemaker 2007), the Trust Region Implementation in Kriging-based optimization with Expected Improvement (TRIKE) (Regis 2016), and the Trust-Region framework for Efficient Global Optimization (TREGO) (Diouane et al. 2023). Although many of these methods confine the search to a local region, the TREGO method alternates between local and global searches, returning to the global scale search after a successful local

search.

In TREGO, a local search is conducted when an iteration fails to achieve meaningful progress, meaning there is no significant improvement over the current best solution. Repeated failures progressively reduce the size of the local search space, while successful iterations trigger a global search. However, this process can lead to slow convergence.

To address this, we propose a modification: successful iterations will continue to focus on a reduced local search space, while unsuccessful iterations will initiate a global search. Here, a successful iteration is defined as one in which a point is found with a lower function value than any previously evaluated. We show that this proposed method leads to faster convergence compared to existing techniques, particularly TREGO.

4.2.2 The Efficient Global Optimization (EGO) Algorithm

The Efficient Global Optimization (EGO) algorithm builds upon the Expected Improvement (EI) criterion to sequentially explore the objective function. It begins with an initial design (typically a Latin hypercube) and iteratively visits the global maximizer of the EI criterion. At each step, the Kriging surrogate model is updated, including the re-estimation of hyperparameters, based on newly sampled points. This process continues until a convergence criterion is met or the budget of function evaluations is exhausted (Roustant, Ginsbourger, and Deville 2012). Algorithm 2 summarizes the procedure.

The EGO algorithm has proven to be highly effective and is frequently used in optimization problems where the number of objective function evaluations is severely limited. It is considered one of the reference methods for global optimization in moderate-dimensional problems (typically $d \leq 10$) (Jones 2001).

Algorithm 2 EGO algorithm

Require: $\boldsymbol{X}$, f = function to be minimized, n_{new}=number of points to add

1: Evaluate f at the design points $\boldsymbol{X}$; $\boldsymbol{y} = f(\boldsymbol{X})$
2: Build a kriging model based on $\boldsymbol{X}$ and $\boldsymbol{y}$
3: **for** i in 1 to n_{new} **do**
4: Find $\boldsymbol{x}^* \leftarrow \arg\max_{\boldsymbol{x}} \mathrm{E}[I(\boldsymbol{x})]$
5: Evaluate $y^* \leftarrow f(\boldsymbol{x}^*)$
6: Update $\boldsymbol{X}$ and $\boldsymbol{y}$ with the new point $\boldsymbol{x}^*$ and response y^*
7: Update the kriging model
8: **end for**
9: Return $\boldsymbol{X}, \boldsymbol{y}$

4.2.3 The Trust Region EGO (TREGO)

Proposed by **diouane2022trego**, Trust Region Efficient Global Optimization (TREGO) operates by searching within a dynamically adjusted trust region centered at the current best solution. The algorithm identifies new candidate points by optimizing within this region and evaluates the objective function. If the result is a success, the region expands, controlled by α. An iteration is deemed successful when there is a significant improvement on the result obtained from the current result by a margin. On the other hand, if poor, it contracts, controlled by β. After each evaluation, the surrogate model is updated with the new data, and the trust region is resized accordingly. This process balances global exploration and local refinement until stopping criteria are met. The stopping criteria mostly used is the maximum iterations. The default values used for α and β are 1/0.9 and 0.9 respectively. Algorithm 3 summarizes the procedure. A forcing function is used to determine whether a search is deemed successful or not. The forcing function $\rho(\sigma)$ is a positive continuous nondecreasing function such that $\rho(\sigma) \to 0$ when $\sigma \to 0$.

Algorithm 3 Simplified Trust-Region EGO (TREGO)

Input: Initial DoE $\boldsymbol{X}_0$, function evaluations $\boldsymbol{y}_0$, constants α, β, $d_{\min}$, $d_{\max}$, initial step-size σ_0, initial best point $\boldsymbol{x}_0^* \in \boldsymbol{X}_0$

Output: Best found point x_k^*

Initialize: $k = 0$

while stopping criterion is not met **do**

    Global Phase

    $\boldsymbol{x}_{\text{global}} = \underset{\boldsymbol{x}\in\Omega}{\operatorname{argmax}}\, \mathbb{E}\left(\mathrm{I}(\boldsymbol{x})\right)$ */* Find global candidate */*

    Update $\boldsymbol{X}_{k+1} := \boldsymbol{X}_k \cup \boldsymbol{x}_k^{\text{global}}$ and $y_{k+1} := \boldsymbol{y}_k \cup f(\boldsymbol{x}_k^{\text{global}})$

    if $f(\boldsymbol{x}_{\text{global}}) \leq f(\boldsymbol{x}_k^*) - \rho(\sigma_k)$: **then** */* Successful global step */*

        Update best point: $\boldsymbol{x}_{k+1}^* = x_{\text{global}}$

        Increase step size: $\sigma_{k+1} = \alpha\sigma_k$

    else

        Local Phase

        Define trust region $\Omega_k = \{\boldsymbol{x} \in \Omega \mid d_{\min}\sigma_k \leq \|\boldsymbol{x} - \boldsymbol{x}_k^*\| \leq d_{\max}\sigma_k\}$

        $\boldsymbol{x}_{\text{local}} = \underset{\boldsymbol{x}\in\Omega_k}{\operatorname{argmax}}\, \mathbb{E}\left(\mathrm{I}(\boldsymbol{x})\right)$ */* Find local candidate */*

        Update $\boldsymbol{X}_{k+1} := \boldsymbol{X}_k \cup \boldsymbol{x}_k^{\text{local}}$ and $y_{k+1} := \boldsymbol{y}_k \cup f(\boldsymbol{x}_k^{\text{local}})$

        if $f(\boldsymbol{x}_{\text{local}}) \leq f(\boldsymbol{x}_k^*) - \rho(\sigma_k)$: **then** */* Successful local step */*

            Update best point: $\boldsymbol{x}_{k+1}^* = \boldsymbol{x}_{\text{local}}$

            Increase step size: $\sigma_{k+1} = \alpha\sigma_k$

        else

            Keep current best point: $\boldsymbol{x}_{k+1}^* = \boldsymbol{x}_k^*$

            Reduce step size: $\sigma_{k+1} = \beta\sigma_k$

        end if

    end if

    Increment iteration counter: $k = k + 1$

end while

    return Best found point $\boldsymbol{x}_k^*$

4.3 The Proposed Algorithm

One key feature of the Efficient Global Optimization (EGO) algorithm is its ability to balance exploration and exploitation to efficiently search the solution space. By exploring broadly, EGO can uncover regions that are more likely to contain the global optimum, which is especially useful when dealing with complex and multi-modal objective functions. However, this global search strategy can be computationally expensive, particularly when the search space is large or evaluating the objective function is time-consuming. In some cases, a localized search focusing on a smaller region may be more efficient.

To address these challenges, we propose a sequential region shrinkage method that alternates between a localized search and a global search. The global search evaluates whether further improvements can be made beyond the current local search region. This approach aims to strike a balance between efficient local searches and the broader exploration required for finding the global optimum. Algorithm 4 summarizes the procedure.

4.3.1 Region of Interest (ROI) Determination

In this method, a controlled parameter ρ determines the proportion of the top $100\rho\%$ of data points to use in calculating the new ROI. In the first iteration, the entire domain Ω serves as the initial ROI. EGO adds n_{new} points to the data set $\boldsymbol{X}$ and $\boldsymbol{y}$, constituting a global search. Afterward, the top $100\rho\%$ of data points, based on a predefined ρ value, are selected. The ROI's boundaries are defined as the minimum and maximum of each dimension from these selected points. The ROI is then shifted to center it at the best-performing point, focusing the subsequent local search on this smaller region.

Figure 4.1 illustrates the ROI determination process.

In Figure 4.1(a), the contours represent the entire function domain Ω. We start with 10 points, selected using a maximin Latin hypercube design. These are represented as the black dots in Figure 4.1. Then 5 additional points are added using EGO. These additional points

Algorithm 4 Sequential Region Shrinkage Method

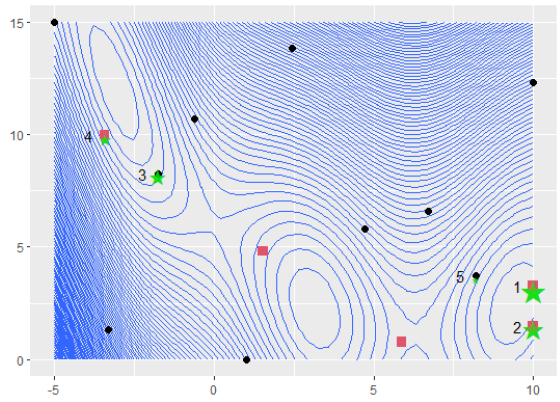
Require: $\boldsymbol{X}$, f, Ω = domain, n_{new}, ρ

1: Evaluate f at the design points $\boldsymbol{X}$; $\boldsymbol{y} \leftarrow f(\boldsymbol{X})$
2: Build a Kriging model based on $\boldsymbol{X}$ and $\boldsymbol{y}$
3: Set the region of interest (ROI) $\Omega' \leftarrow \Omega$ (initial domain)
4: **while** stopping criteria not met **do**
5: Run EGO within the domain Ω' to obtain the next n_{new} points
6: Update $\boldsymbol{X}$ and $\boldsymbol{y}$ with the new points
7: Update the Kriging model
8: Determine the new ROI $\Omega' \leftarrow \text{ROI}(\boldsymbol{X}, \boldsymbol{y}, \Omega, \rho)$
9: **if** small or no improvement (unsuccessful iteration) **then**
10: Restore the original domain: $\Omega' \leftarrow \Omega$
11: **end if**
12: **end while**
13: Return $\boldsymbol{X}$, $\boldsymbol{y}$

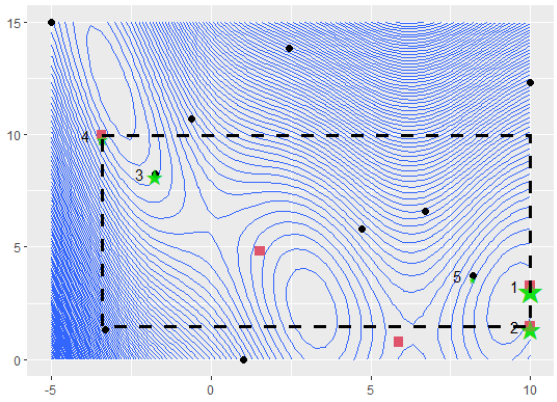
Algorithm 5 Determine Region of Interest (ROI)

Require: $\boldsymbol{X}$, Ω = domain, $\boldsymbol{y}$, ρ.

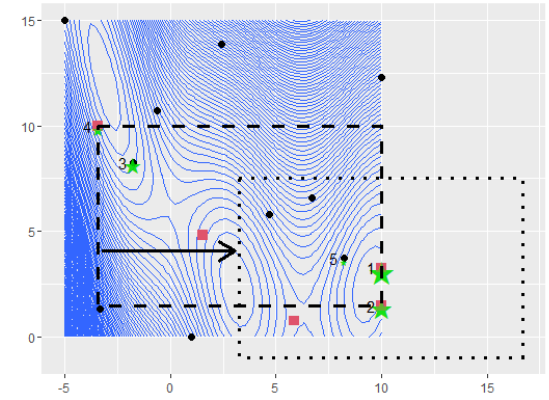
1: $\tau \leftarrow$ indices of the top $100\rho\%$ of $\boldsymbol{y}$.
2: Select points in $\boldsymbol{X}$ associated with τ.
3: Find the best point $\boldsymbol{x}^*$ that minimizes the objective function
4: Determine the lower and upper bounds for points associated with τ; set $L_j := \min_{i \in \tau} x_{ij}$ and $U_j := \max_{i \in \tau} x_{ij}$ for each $j = 1, \ldots, d$
5: Set $\mathbf{D} := (\mathbf{U} - \mathbf{L})/2$, where $\mathbf{U} = (U_1, \ldots, U_d)$ and $\mathbf{L} = (L_1, \ldots, L_d)$.
6: Set $\Omega' = \{\boldsymbol{x}^* - \mathbf{D}, \boldsymbol{x}^* + \mathbf{D}\} \cap \Omega$.
7: **return** Ω'.



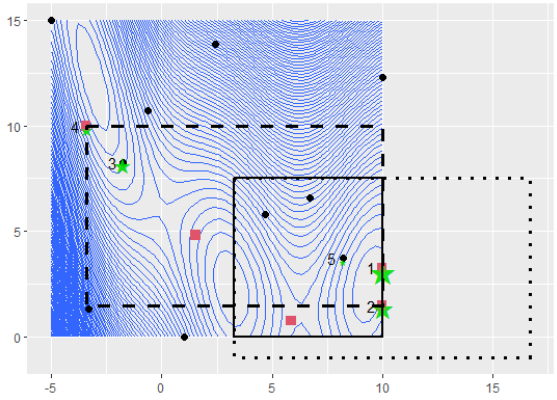
(a) Original function domain Ω with 5 points added using EGO



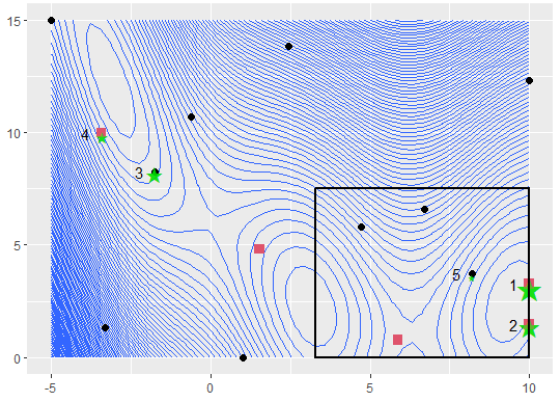
(b) Initial ROI containing the top 30% of points



(c) Centering the ROI at the best point (labeled 1)



(d) Constraining the ROI to the initial domain Ω



(e) ROI to be used for the next iteration

Figure 4.1: ROI determination using the top 30% of total points. The top points are labeled 1-5.

are indicated by the red squares. Taking $\rho = 0.3$, the best 30% of the 15 points, labeled 1-5 and indicated by green stars in decreasing order, are then used to define the new ROI, as shown in Figure 4.1(b). The ROI is centered around the best-performing point (labeled 1) in Figure 4.1(c). Since part of the ROI extends outside the original domain, it is constrained within Ω, resulting in the boxed region in Figure 4.1(d) and 4.1(e). This becomes the new ROI, Ω', for the first iteration.

Next, a local search is performed within the newly defined ROI, and m points are added that maximize the Expected Improvement (EI). With each addition, the Kriging model is updated to reflect the newly sampled points. If a point significantly improves upon the current minimum, the iteration is considered successful, and a new ROI is determined using the updated top $100\rho\%$ of data points. Otherwise, the method reverts to a global search.

This iterative process continues until a predefined stopping criterion is met, such as reaching a specified tolerance level or a maximum number of function evaluations. The stopping criterion can be adapted depending on whether the true optimum is known. If the optimum is known, we stop when the tolerance error is reached. Otherwise, we allow for a default of three global search iterations based on empirical tests, which showed that by the third global search, convergence typically occurs.

The region bounding strategy is designed to leverage the fact that a well-spaced design has a high likelihood of sampling points near the optimal region. By focusing the search in the most promising areas and selectively switching to global searches when necessary, the proposed method efficiently balances exploration and exploitation. As a result, it achieves the optimal solution with fewer function evaluations compared to standard EGO and TREGO algorithms.

4.3.2 Difference between RSO and TREGO

TREGO uses EGO to do optimization within the function domain. Once a a point is obtained whose value is below the current function minimum by a given threshold, the iteration is

considered to be successful. The next optimization is to run EGO in the entire function domain. In the case of an unsuccessful iteration, the region is shrinked by a factor of beta. The shrinked region is then centered in the current function minimum, where EGO is then run. The local optimization is only done after the global iteration is unsuccessful. This is different from the RSO since in the RSO, the region is determined by the top $100\rho\%$ of the data so far used. Also the RSO reverts to the global search after an unsuccessful iteration.

4.4 Numerical Experiments

To illustrate the efficiency of the method, various test functions of various dimensions from the Virtual Library of Simulation Experiments (Surjanovic and Bingham 2013) were taken into consideration. A maximin Latin hypercube design with $n = 5d$ runs was selected as the initial design. The results were compared to those obtained by EGO and TREGO.

The test functions used and their settings are elaborated below:

1. **Branin function** $(d = 2)$

$$f(x_1, x_2) = \left(x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$

with $x_1 \in [-5, 10], x_2 \in [0, 15]$. The global minimum $f(\boldsymbol{x}^*) = 0.397887$ are located at $\boldsymbol{x}^* = (-\pi,\, 12.275), (\pi, 2.275)$ and $(9.42478, 2.475)$.

2. **Six-hump camel function (SixCamel)** $(d = 2)$

$$f(x_1, x_2) = 4x_1^2 - 2.1x_1^4 + x_1^6/3 + x_1x_2 - 4x_2^2 + 4x_2^4$$

with $-2 \leq x_1 \leq 2, -1 \leq x_2 \leq 1$. It has six extreme points, among them the global minimum are $\boldsymbol{x}^* = (0.0898, -0.7126), (-0.0898, 0.7126)$ and $f(\boldsymbol{x}^*) = -1.0316$.

3. **Goldstein-Price function** $(d = 2)$

$$\begin{aligned} f(\mathbf{x}) =& \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right] \\ & \times \left[30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right] \end{aligned}$$

Table 4.1: Parameters for the Hartmann functions

Functions	Parameters
Hartmann3 $\boldsymbol{x}^* = (0.1146, 0.5556, 0.8525)$ $f(\boldsymbol{x}^*) = -3.86278$	$\boldsymbol{A} = \begin{pmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}$; $\boldsymbol{P} = 10^{-4} \begin{pmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{pmatrix}$
Hartmann4 $\boldsymbol{x}^* = (0.1873, 0.1906,$ $0.5566, 0.2647)$ $f(\boldsymbol{x}^*) = -3.135474$	$\boldsymbol{A} = \begin{pmatrix} 10.0 & 0.05 & 3.0 & 17.00 \\ 3.0 & 10.00 & 3.5 & 8.00 \\ 17.0 & 17.00 & 1.7 & 0.05 \\ 3.5 & 0.10 & 10.0 & 10.00 \\ 1.7 & 8.00 & 17.0 & 0.10 \\ 8.0 & 14.00 & 8.0 & 14.00 \end{pmatrix}$; $\boldsymbol{P} = 10^{-4} \begin{pmatrix} 1312 & 2329 & 2348 & 4047 \\ 1696 & 4135 & 1451 & 8828 \\ 5569 & 8307 & 3522 & 8732 \\ 124 & 3736 & 2883 & 5743 \\ 8283 & 1004 & 3047 & 1091 \\ 5886 & 9991 & 6650 & 381 \end{pmatrix}$
Hartmann6 $\boldsymbol{x}^* = (0.2017, 0.1500, 0.4769$ $0.2753, 0.3117, 0.6573)$ $f(\boldsymbol{x}^*) = -3.32237$	$\boldsymbol{A} = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}$ $\boldsymbol{P} = 10^{-4} \begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix}$

with $-2 \leq x_1, x_2 \leq 2$. The global minimum $f(\boldsymbol{x}^*) = 3$ is at point $\boldsymbol{x}^* = (0, -1)$ with several local minima around it.

4. **Hartmann functions** with $d = 3$ (Hartmann3), $d = 4$ (Hartmann4) and $d = 6$ (Hartmann6)

$$f(\boldsymbol{x}) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{d} A_{ij} \left(x_j - P_{ij}\right)^2\right),$$

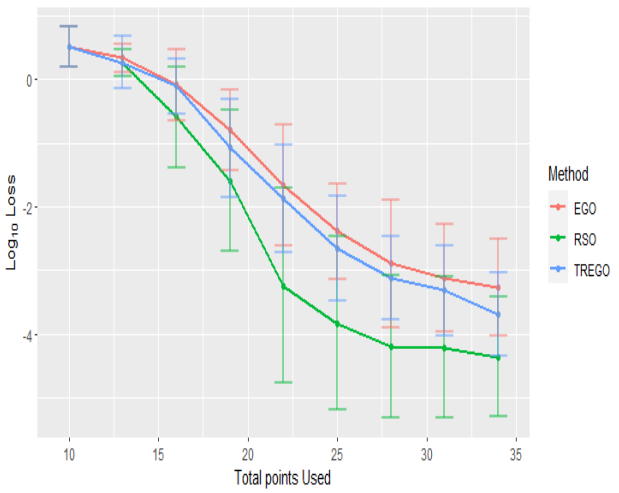
where $0 \leq x_i \leq 1, i = 1, 2, \cdots, d$ and $\alpha = (1.0, 1.2, 3.0, 3.2)^T$. The elements of matrices $\boldsymbol{A}, \boldsymbol{P}$ and the global minimum are given in Table 4.1.

Table 4.2: Number of function evaluations to achieve the desired tolerance level

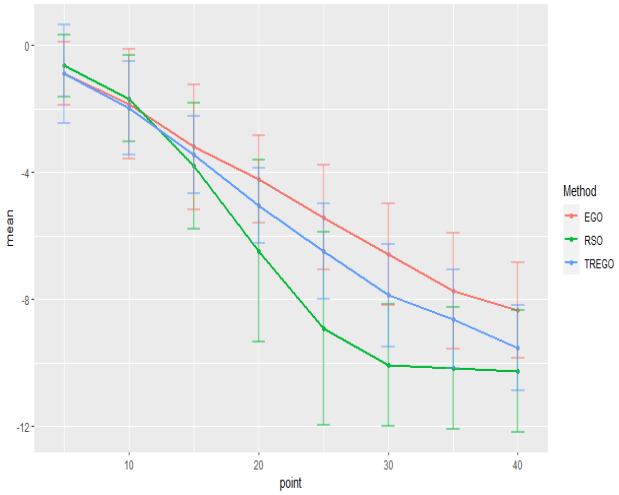
		mean			median		
Function	tolerr	EGO	TREGO	RSO	EGO	TREGO	RSO
Branin	10^{-3}	35.75	28.75	26.25	35	30	25
	10^{-4}	42	37.25	32.5	40	40	30
SixCamel	10^{-3}	43.50	41.00	33.75	45	40	35
	10^{-4}	48.75	46.75	40.75	50	50	40
Goldstein-Price	10^{-3}	67	52.5	41	70	57.5	35
	10^{-4}	70	70	42.5	70	70	37.5
Hartmann3	10^{-3}	23.4	21.2	20.8	24	20	20
	10^{-4}	30.8	25.75	26.4	26	22	24
Hartmann4	10^{-3}	46.0	44.0	40.0	44	42	36
	10^{-4}	63.6	55.2	53.6	46	44	40
Hartmann6	10^{-3}	60.5	58.5	49.5	64	61	42.5
	10^{-4}	67.5	62	52	70	67.5	47.5

Functions with dimensions 6 or less were considered to be low dimensional functions. In the optimization, we used error between the function value evaluated at the best point and the known global optimum. We were interested in the number of function evaluations used by the algorithm to converge. This was replicated 20 times. The results were compared to the ones obtained when running EGO alone and when using TREGO. The results are reported in Table 4.2.

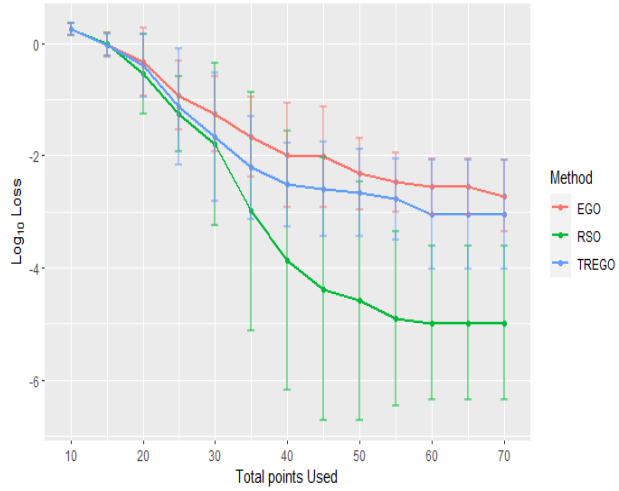
From the results above, we note that the proposed method converges with fewer function evaluations than both EGO and TREGO in all but one instances. This efficiency not



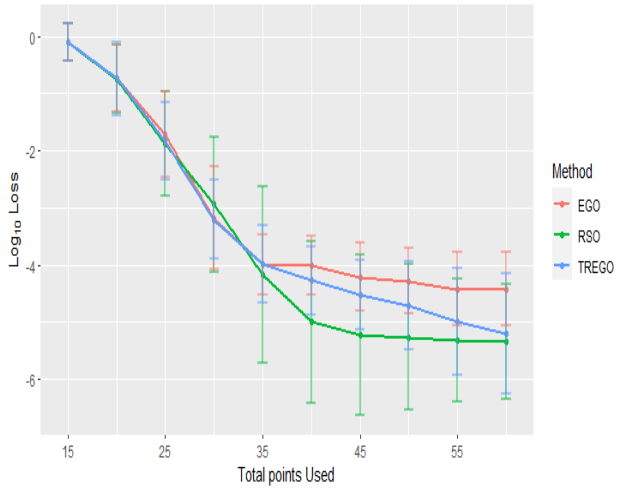
(a) Branin function



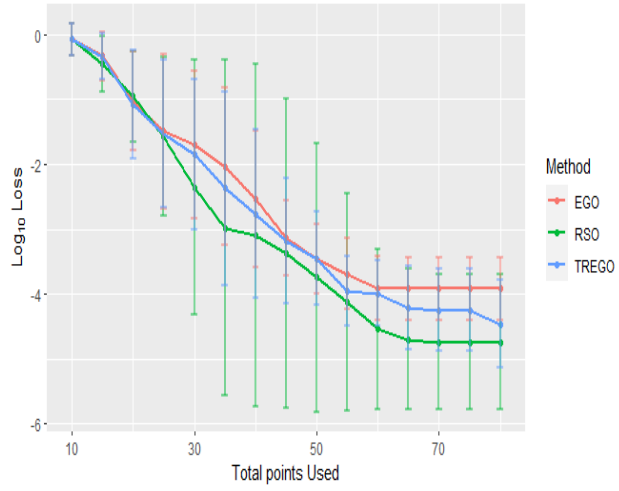
(b) SixCamel function



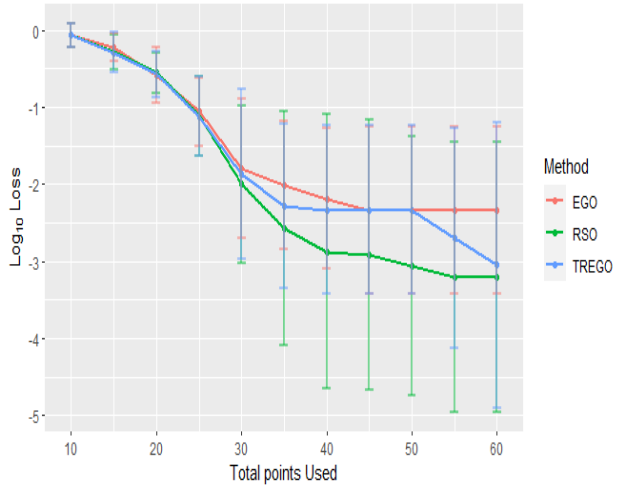
(c) Goldstein-Price function



(d) Hartmann3 function



(e) Hartmann4 function



(f) Hartmann6 function

Figure 4.2: Comparison of 3 methods for different test functions

only conserves computational resources but also is particularly beneficial in resource-limited scenarios. The exploitation characteristics of our method contribute to this performance. Notably, while our method generally outperform EGO and TREGO in all instances, for the Hartmann3 function and tolerance of 10^{-4}, TREGO performs slightly better than our method.

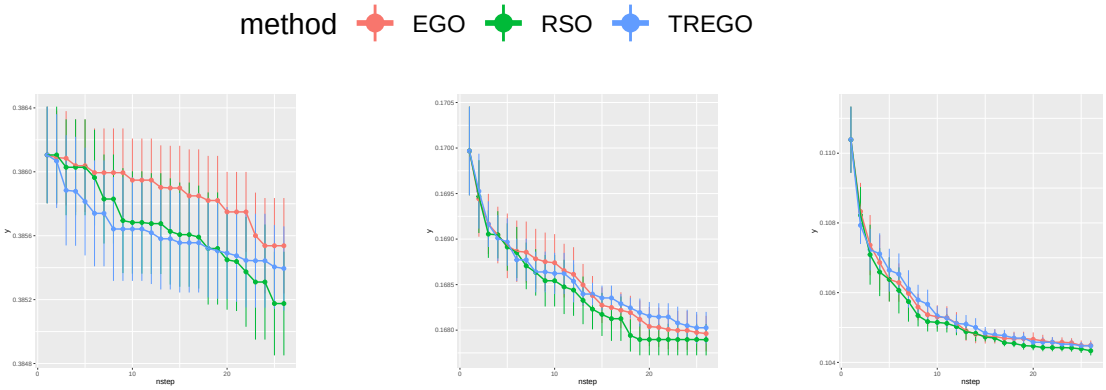
Additionally, Figure 4.2 shows the minimization path for different test functions using the 3 methods, along with the 2 standard error bars at each step of the minimization path. The log mean error analysis indicates that the proposed method rapidly decreases the error rate before stabilizing, signifying a faster convergence compared to the alternatives. In constrained resource settings, where only 15-25 new observations can be made, our method achieves a log10 error of at most -3 across all tested functions.

4.5 Application in Generating Uniform Projection Designs

We consider tuning the DE algorithm from UniPro package to generate uniform projection designs (UPDs). As the DE algorithm is stochastic, the generation is replicated 10 times and the mean of the 10 criterion values obtained. This is done to minimize variation. This is considered to be the objective function to be optimized.

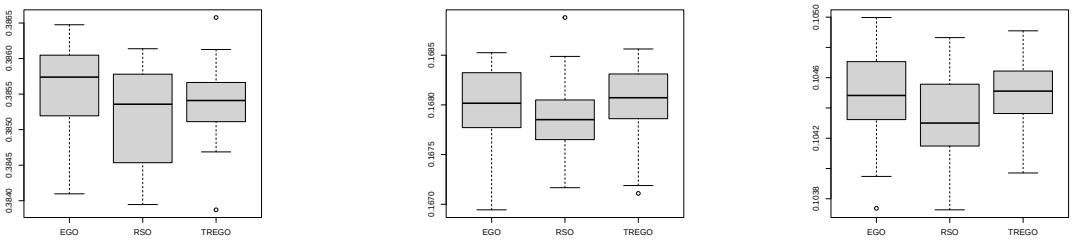
As described in Chapter 2, the DE algorithm has five hyperparameters: NP, itermax, pCR, pMut and pGBest. We use EGO, TREGO and RSO to tune DE hyperparameters. Starting with a random Latin hypercube of size $5d = 25$, we obtain the minimization path by obtaining 25 additional points. This is replicated 20 times. Figure 4.3 shows the minimization path and Figure 4.4 shows the boxplots of the final optimal objective values for three methods. The RSO method tends to generally provide smaller objective values than the EGO or TREGO method.

We also take a look at the distribution of the optimal hyperparameter settings from the 20 replicates in Figure 4.5 obtained from the RSO method, where NP and itermax are



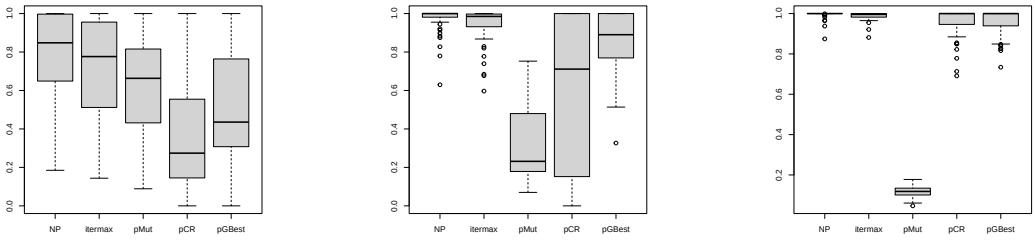
(a) 30×3 (b) 50×5 (c) 70×7

Figure 4.3: Minimization path for 3 methods in the construction of UPDs using DE



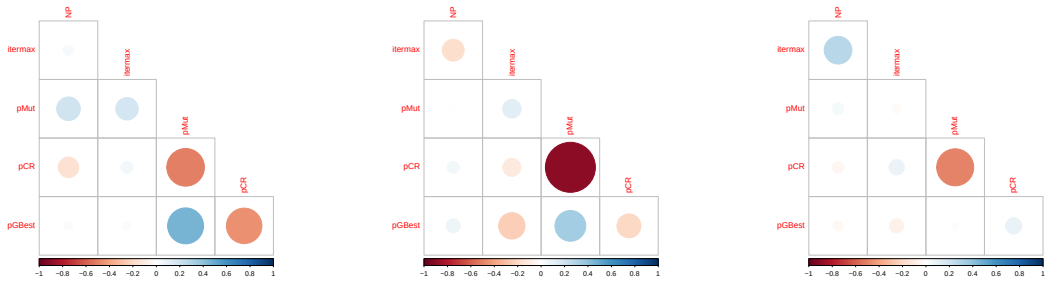
(a) 30×3 (b) 50×5 (c) 70×7

Figure 4.4: Comparison of optimal results from 3 methods



(a) 30×3 (b) 50×5 (c) 70×7

Figure 4.5: Distribution of optimal hyperparameter settings



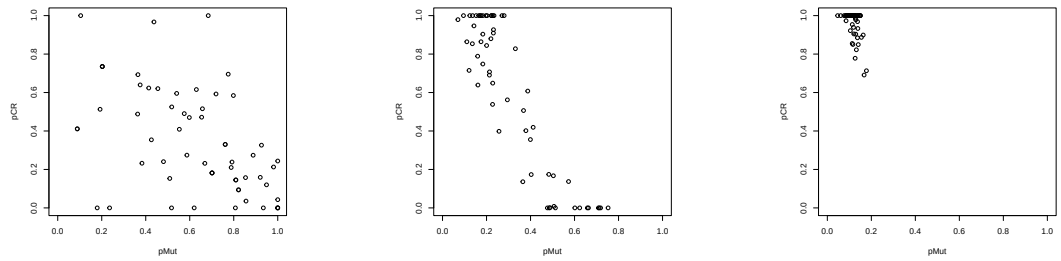
(a) 30×3 (b) 50×5 (c) 70×7

Figure 4.6: Correlation plot of the optimal hyperparameter settings

rescaled to $[0, 1]$. This determines as to wether the optimal region is the same point or we are converging to a local optimum. The 30×3 target design indicates that the process does not converge to the same value each time. On the other hand, from the 50×5 and 70×7 target designs, we see that the values for NP and itermax are concentrated at the upper vertex. Finally, from the 70×7 target design, we find out that the values for pCR and pGBest are also concentrated at the upper end. However, none of the target designs give information about where the probability of mutation (pMut) should lie. We examine the correlations to determine as to whether there exists a two way correlation amongst the parameters.

From the correlation plots in Figure 4.6, we do notice that pMut is highly correlated to pCR. These two hyperparameters could be optimized simultaneously while holding the other 3 hyperparameters (NP, itermax, pGBest) at their highest level. The justification for this is due to the fact that NP and itermax could be thought of as the budget size. For the probability of using global best, this value should be high since we would want to be as close as possible to the global minimum. Figure 4.7 shows the scatter plots of pMut and pCR. These two hyperparameters are intriguing as they are highly negatively correlated.

Having set these three at their maximum level, an optimization is carried out on the two remaining parameters, pMut and pCR. The results obtained do not differ from the 5 hyperparameter optimization, though takes a bit longer since NP and itermax have been held at their highest level.



(a) 30×3 (b) 50×5 (c) 70×7

Figure 4.7: Scatter plots showing the relation between pMut and pCR

Finally, we compare the RSO method with other methods in generating UPDs. We consider the random search method, the grid search method, the DE1 and DE4 methods provided by Stokes, Wong, and Xu (2024), and the DEoptim method based on the second order model in Chapter 2. The random search used a random Latin hypercube design with 1024 runs and the grid search method used a 4^5 full factorial design. The optimal factor combinations obtained for the random search and grid search are given in Tables 4.3 and 4.4, respectively. DE1 always uses the global best (i.e., pGBest=1), while DE4 is a hybrid version with pGBest=0.5, which randomly chooses the global best, the current agent and a random agent with probability 50%, 25%, and 25% for each column independently. Both DE1 and DE4 use NP = 100, itermax = 1500, pMut = 0.1 and pCR = 0.5. The DEoptim method from Chapter 2 uses NP = 100, itermax = 1500, pGBest = 0.95 and optimal settings of (pMut, pCR) as (0.95, 0.05), (0.25, 0.75), and (0.15, 0.85), respectively, for target size 30×3, 50×5 and 70×7. Table 4.5 gives the optimal hyperparameter settings obtained from the RSO method.

For a given target size and each method, we generated 100 UPDs using the respective optimal factor combination and computed their objective $\phi(\cdot)$ values. Figure 4.8 compares the objective values of the generated UPDs from the six methods.

Figure 4.8 indicates the advantage of using optimized methods as compared to random search or grid search. Even using 1024-runs to determine the optimal settings, the grid and

Table 4.3: Optimal hyperparameter settings for each target design with 4^5 LHD

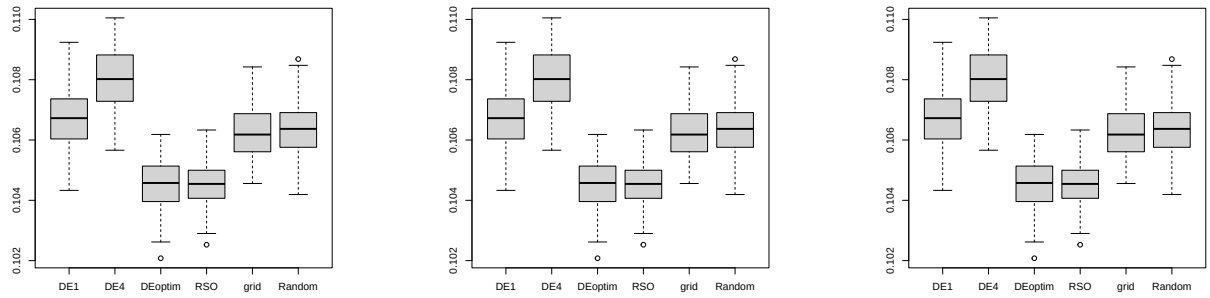
	NP	itermax	pMut	pCR	pGBest	$\phi(\cdot)$
30×3	95	917	0.63	0.19	0.72	0.3850
50×5	98	1485	0.40	0.38	0.88	0.1685
70×7	84	1408	0.11	0.92	0.86	0.1066

Table 4.4: Optimal hyperparameter settings for each target design with 4^5 FFD

	NP	itermax	pMut	pCR	pGBest	$\phi(\cdot)$
30×3	100	1167	0.95	0.05	0.65	0.3845
50×5	100	1500	0.35	0.95	0.95	0.1684
70×7	100	1500	0.35	0.35	0.95	0.1055

Table 4.5: Optimal hyperparameter settings for each target design using RSO

	NP	itermax	pMut	pCR	pGBest
30×3	100	1452.24707728029	0.95	0.05	0.737833018147394
50×5	100	1500	0.194770169523594	0.625050426917319	0.736810635174503
70×7	100	1498.2370130889	0.125822708310132	0.948123528142478	0.95



(a) 30×3 as target (b) 50×5 as target (c) 70×7 as target

Figure 4.8: Comparison of six methods for constructing UPDs

random search methods perform poorly as compared to RSO and DEoptim. On the other hand, DE1 and DE4 perform the worst as these methods use arbitrary settings selected by Stokes, Wong, and Xu (2024).

Figure 4.8 also indicates that using the RSO as compared to the default DE1 and DE4 is highly effective when the dimension of the target design increases. This is because with increase in dimension of target design, the DE1/DE4 algorithms are still far from the optimal solution, while the RSO carries out an additional minimization which moves closer to the optimal hyperparameter settings. In addition, RSO and DEoptim have similar performance despite the use of different settings for pCR, pMut and pGBest. One advantage of using RSO over DEoptim is the notion that for DEoptim, one has to determine which design is best, to carry out the optimization. In Chapter 2, it is shown that the 50-run OACD is the best. On the other hand, RSO uses a 25-run LHD with additional 25 points for optimization. No prior knowledge of the design is needed. In addition, determining the optimal settings in DEoptim is quite a task when there exists significant interactions between the factors, while for RSO, the method yields the optimal settings directly.

CHAPTER 5

Conclusion and Future Directions

The studies presented in this work advance the understanding and application of optimization methods and experimental design in addressing complex, high-dimensional challenges in design construction. Optimization algorithms, especially Differential Evolution (DE) and Bayesian optimization, have demonstrated their capability to tackle intricate real-world problems requiring efficient navigation of extensive search spaces. This dissertation has focused on three interlinked studies: optimizing DE hyperparameters for generating Uniform Projection Designs (UPDs), evaluating initial design choices for Efficient Global Optimization (EGO), and introducing a Kriging-based sequential shrinkage method for enhancing Bayesian optimization. Collectively, these studies provide essential insights into how the structure and strategy of optimization methods can be tailored to suit varied problem domains, resulting in improved accuracy, efficiency, and scalability.

This chapter synthesizes the findings of these studies, discussing their implications and highlighting directions for future work in advancing optimization techniques.

5.1 Key Findings Across the Studies

5.1.1 Optimizing Hyperparameters in Differential Evolution for UPD Generation

The first study explored the hyperparameter landscape of a modified DE algorithm applied to discrete tasks, such as generating Uniform Projection Designs (UPDs). DE is traditionally

effective for continuous optimization, but modifications to its structure are necessary to handle discrete variables. Inspired by recent work on DE adaptations, this study focused on five critical DE hyperparameters, including population size, maximum iterations, and probabilities related to mutation, crossover and use of the global best. Through extensive testing, OACD and CCD emerged as the most effective experimental designs for exploring DE's response surface, outperforming space-filling methods in terms of root mean squared error and correlation with the underlying objective.

This analysis, built upon second-order and surrogate models, revealed that second-order models offer a reliable yet simpler alternative to more complex surrogate approaches, such as Kriging and Gaussian Processes (GPs). This is significant because identifying optimal hyperparameter settings is not only crucial for enhancing DE's performance but also for constructing high-quality UPDs, which have applications in a variety of fields that require uniformity across high-dimensional spaces. The study highlights the importance of structured hyperparameter tuning in optimizing DE's utility, establishing OACD and CCD as powerful tools for practitioners seeking to leverage DE in discrete optimization contexts.

5.1.2 Efficiency and Robustness of UPDs in High-Dimensional Prediction and Optimization Tasks

The second study assessed the effectiveness of various initial design strategies within the framework of prediction and optimization. By using Gaussian Process (GP) models as surrogates and focusing on active learning through selective sampling, this study highlighted the influence of initial design on prediction accuracy and optimization efficiency. In particular, Uniform Projection Designs (UPDs) consistently outperformed distance-based designs, such as maximin designs, especially in high-dimensional spaces. This advantage is attributed to the UPD's focus on two-dimensional uniformity via the centered L_2-discrepancy criterion, which circumvents the inefficiencies associated with Euclidean distance metrics in high-dimensional contexts.

The performance of UPDs highlights the limitations of distance-based designs in high dimensions. Euclidean-based metrics, such as those used in maximin designs, tend to falter in high-dimensional settings where points become uniformly distant from one another. In contrast, UPDs maintain efficiency by avoiding reliance on high-dimensional distances, instead leveraging two-dimensional uniformity, which ensures more consistent prediction accuracy across varying dimensionalities. This study's findings underscore the importance of initial design choice in prediction and also in optimization via EGO processes, particularly when early-stage optimization has a pronounced impact on convergence and overall accuracy.

5.1.3 Introducing a Sequential Shrinking Approach to Bayesian Optimization for Resource-Efficient Tuning

The third study introduced a Kriging-based sequential region shrinking method, enhancing traditional Bayesian optimization techniques by focusing on progressively smaller, more promising regions in the search space. This novel approach incorporates aspects of EGO by iteratively narrowing the search region, allowing the optimization process to converge more efficiently. By targeting high-potential areas in each iteration, the approach significantly reduces the computational resources required compared to traditional grid and random search methods.

The sequential shrinking method not only performs effectively across various test functions but also demonstrates advantages over existing Bayesian strategies, such as TREGO. This is particularly important in resource-constrained scenarios, where reducing computational load without sacrificing optimization quality is paramount. The method's integration of Kriging with a focused region-based approach makes it a practical choice for applications in high-dimensional spaces, where traditional optimization strategies may become computationally prohibitive. As such, this study's findings contribute to a growing body of work emphasizing the role of adaptive, resource-efficient techniques in optimization.

5.2 Implications for Optimization Strategies and Practical Applications

This dissertation's findings emphasize the role of both design selection and adaptive optimization strategies in achieving high performance in complex, high-dimensional tasks. The successful use of UPDs as initial designs in EGO demonstrates that high-dimensional optimization can be more effectively approached by leveraging two-dimensional uniformity rather than high-dimensional Euclidean distances. This insight is essential for practitioners in fields where high-dimensional optimization is necessary, offering a robust alternative to conventional designs that suffer in high-dimensional spaces.

Moreover, the effectiveness of structured search techniques, particularly the novel Kriging-based sequential region shrinking method, highlights the potential of region-focused optimization in resource-constrained applications. By progressively shrinking the search region, this method achieves convergence with reduced computational requirements, which has practical implications for industries where efficiency and speed are critical. Similarly, the use of OACD and CCD in hyperparameter tuning underscores the continued relevance of classical factorial composite designs and the response surface methodologies, particularly when dealing with highly sensitive or discrete parameters in algorithms like DE.

For practitioners, these studies provide practical insights into choosing appropriate design and search techniques based on problem dimensionality, optimization objectives, and computational constraints. The findings also underscore the value of adaptive frameworks in both design and search, offering a balance between exploration and exploitation that improves optimization outcomes.

5.3 Limitations and Future Directions

Despite the strengths of the findings, this research also identifies limitations that warrant further investigation. First, while the current studies provide evidence for the effectiveness of

UPDs in high-dimensional optimization, additional work is needed to explore the theoretical underpinnings of UPDs' performance.The impact of factor interactions in the DE algorithm remains an area for future exploration, particularly in developing more nuanced models that account for interaction effects without introducing excessive model complexity.

Additionally, the sequential shrinking approach, while efficient, could benefit from integration with TREGO or with other advanced machine learning models or ensemble methods, potentially enhancing its adaptability and performance in even more complex search spaces. For instance, exploring hybrid approaches that combine GP models with UPD-based designs or integrating deep learning models for dynamic surrogate modeling may further improve accuracy and computational efficiency.

Furthermore, the Kriging-based shrinking approach and DE hyperparameter optimization would benefit from more comprehensive evaluations across different domains. Future work might explore alternative acquisition functions that enhance adaptability in Bayesian optimization, especially for applications requiring specific trade-offs between exploration and exploitation. Extending these techniques to larger datasets or real-world applications would also validate their utility in practical settings, where high dimensionality and computational costs pose ongoing challenges.

Finally, this research although focuses exclusively on the construction of Uniform Projection Designs (UPDs), it is generalizable to the construction of various space filling designs. We considered the UPD as there is no established construction methods that currently exist. In future studies, however, we can apply the proposed procedures to construct other space filling desings such as maxPro designs or uniform designs and then compare these with existing designs to assess the efficiency of the proposed methods.

5.4 Final Remarks

In conclusion, this work advances the field of high-dimensional optimization by evaluating and enhancing design and optimization strategies for hyperparameter tuning, experimental design generation, and adaptive region-focused optimization. The findings from the three studies provide a cohesive framework that balances theoretical insights with practical applicability, ultimately paving the way for more robust and efficient optimization techniques. This research not only enhances our understanding of the design and optimization landscape but also contributes valuable tools for tackling complex challenges across scientific, engineering, and industrial applications. The evolving methodologies and insights presented here have significant potential to guide future innovations, leading to more accurate, scalable, and resource-efficient solutions for a wide array of optimization problems.

References

Aarts, Emile and Jan Korst (1989). *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc.

Adorio, Ernesto P and U Diliman (2005). "Mvf-multivariate test functions library in c for unconstrained global optimization". In: *Quezon City, Metro Manila, Philippines* 44.

Aggarwal, Charu C, Alexander Hinneburg, and Daniel A Keim (2001). "On the surprising behavior of distance metrics in high dimensional space". In: *Database theory—ICDT 2001: 8th international conference London, UK, January 4–6, 2001 proceedings 8*. Springer, pp. 420–434.

Babu, BV and M Mathew Leenus Jehan (2003). "Differential evolution for multi-objective optimization". In: *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* Vol. 4. IEEE, pp. 2696–2703.

Bergstra, James and Yoshua Bengio (2012). "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2.

Beyer, H-G and Dirk V Arnold (2001). *Theory of evolution strategies-A tutorial*. Springer.

Beyer, Hans-Georg and Hans-Paul Schwefel (2002). "Evolution strategies–a comprehensive introduction". In: *Natural computing* 1, pp. 3–52.

Binois, Mickael, Robert B Gramacy, and Mike Ludkovski (2018). "Practical heteroscedastic gaussian process modeling for large simulation experiments". In: *Journal of Computational and Graphical Statistics* 27.4, pp. 808–821.

Blum, Christian (2005). "Ant colony optimization: Introduction and recent trends". In: *Physics of Life reviews* 2.4, pp. 353–373.

Box, George E P and Kenneth B Wilson (1951). "On experimental attainment of optimum conditions". In: *Journal of the Royal Statistical Society, Series B* 13.1, pp. 1–45.

Brochu, Eric, Vlad M Cora, and Nando De Freitas (2010). "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". In: *arXiv preprint arXiv:1012.2599*.

Burton, Henry, Hongquan Xu, and Zhengxiang Yi (2022). "Design of computer experiments for developing seismic surrogate models". In: *Earthquake Spectra* 38.1, pp. 384–406.

Chen, Hao et al. (2016). "Analysis methods for computer experiments: How to assess and what counts?" In: *Statistical Science* 31.1, pp. 40–60.

Chevalier, Clément, Victor Picheny, and David Ginsbourger (2014). "KrigInv: An efficient and user-friendly implementation of batch-sequential inversion strategies based on kriging". In: *Computational statistics & data analysis* 71, pp. 1021–1034.

Clerc, Maurice and James Kennedy (2002). "The particle swarm-explosion, stability, and convergence in a multidimensional complex space". In: *IEEE transactions on Evolutionary Computation* 6.1, pp. 58–73.

Coello, Carlos A Coello, Gregorio Toscano Pulido, and Maximino Salazar Lechuga (2004). "Handling multiple objectives with particle swarm optimization". In: *IEEE Transactions on evolutionary computation* 8.3, pp. 256–279.

Currin, Carla et al. (1991). "Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments". In: *Journal of the American Statistical Association* 86.416, pp. 953–963.

Dahal, Laxman, Henry Burton, and Samuel Onyambu (2022). "Quantifying the effect of probability model misspecification in seismic collapse risk assessment". In: *Structural Safety* 96, p. 102185.

Das, Swagatam and Ponnuthurai Nagaratnam Suganthan (2010). "Differential evolution: A survey of the state-of-the-art". In: *IEEE transactions on evolutionary computation* 15.1, pp. 4–31.

Diouane, Youssef et al. (2023). "TREGO: a trust-region framework for efficient global optimization". In: *Journal of Global Optimization* 86.1, pp. 1–23.

Dixon, Laurence Charles Ward and Giorgio P Szegö (1978). *Towards Global Optimisation 2*. North-Holland Pub. Co.

Domingos, Pedro (2012). "A few useful things to know about machine learning". In: *Communications of the ACM* 55, pp. 78–87.

Dorigo, M (1996). "The Ant System: Optimazation by a colony of cooperation agents". In: *IEEE Trans. Systems, Man and Cybernetics Part B* 26.1, p. 113.

Dorigo, Marco (2007). "Ant colony optimization". In: *Scholarpedia* 2.3, p. 1461.

Falkner, Stefan, Aaron Klein, and Frank Hutter (2018). "BOHB: Robust and efficient hyperparameter optimization at scale". In: *International conference on machine learning*. PMLR, pp. 1437–1446.

Fang, Kai-Tai et al. (2000). "Uniform design: theory and application". In: *Technometrics* 42.3, pp. 237–248.

Feurer, Matthias and Frank Hutter (2019). "Hyperparameter optimization". In: *Automated machine learning: Methods, systems, challenges*, pp. 3–33.

Finney, D. J. (1945). "The Fractional Replication of Factorial Arrangements". In: *Annals of Eugenics* 12, pp. 291–301.

Fisher, R. A. (1935). *The Design of Experiments*. Oliver and Boyd.

Gardner, Martha et al. (2006). "From small X to large X: Assessment of space-filling criteria for the design and analysis of computer experiments". In: *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, p. 1714.

Giunta, Anthony, Steven Wojtkiewicz, and Michael Eldred (2003). "Overview of modern design of experiments methods for computational simulations". In: *41st Aerospace Sciences Meeting and Exhibit*, p. 649.

Hajek, Bruce (1988). "Cooling schedules for optimal annealing". In: *Mathematics of operations research* 13.2, pp. 311–329.

Hansen, Nikolaus and Andreas Ostermeier (2001). "Completely derandomized self-adaptation in evolution strategies". In: *Evolutionary computation* 9.2, pp. 159–195.

Harari, Ofir and David M Steinberg (2014). "Optimal designs for Gaussian process models via spectral decomposition". In: *Journal of Statistical Planning and Inference* 154, pp. 87–101.

Holland, JH (1975). "An introductory analysis with applications to biology, control, and artificial intelligence". In: *Adaptation in Natural and Artificial Systems. First Edition, The University of Michigan, USA*.

Hutter, Frank, Holger H Hoos, and Kevin Leyton-Brown (2011). "Sequential model-based optimization for general algorithm configuration". In: *Learning and Intelligent Optimization: 5th International Conference*. Springer, pp. 507–523.

Johnson, David S et al. (1991). "Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning". In: *Operations research* 39.3, pp. 378–406.

Johnson, Mark E, Leslie M Moore, and Donald Ylvisaker (1990). "Minimax and maximin distance designs". In: *Journal of statistical planning and inference* 26.2, pp. 131–148.

Jones, Donald R (2001). "A taxonomy of global optimization methods based on response surfaces". In: *Journal of global optimization* 21, pp. 345–383.

Jones, Donald R, Matthias Schonlau, and William J Welch (1998). "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13, pp. 455–492.

Jong Kenneth, Alan de (1975). "Analysis of the behavior of a class of genetic adaptive systems". In: *Technical Report No. 185, Department of Computer and Communication Sciences, University of Michigan*.

Joseph, V Roshan, Evren Gul, and Shan Ba (2015). "Maximum projection designs for computer experiments". In: *Biometrika* 102.2, pp. 371–380.

Kawachi, Masahiro and Nobuyoshi Ando (1992). "Goldberg, DE: Genetic Algorithms in Search, Optimization & Machine Learning, 401pp., Addison-Wesley (1989)." In: *Artificial Intelligence* 7.1, pp. 168–168.

Kennedy, James and Russell Eberhart (1995). "Particle swarm optimization". In: *Proceedings of ICNN'95-international conference on neural networks*. Vol. 4. ieee, pp. 1942–1948.

Kirkpatrick, Scott, C Daniel Gelatt Jr, and Mario P Vecchi (1983). "Optimization by simulated annealing". In: *science* 220.4598, pp. 671–680.

Krige, Daniel G (1951). "A statistical approach to some basic mine valuation problems on the Witwatersrand". In: *Journal of the Southern African Institute of Mining and Metallurgy* 52.6, pp. 119–139.

Kukkonen, Saku and Jouni Lampinen (2006). "Constrained real-parameter optimization with generalized differential evolution". In: *2006 IEEE International Conference on Evolutionary Computation*. IEEE, pp. 207–214.

Li, Lisha et al. (2018). "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *Journal of Machine Learning Research* 18.185, pp. 1–52.

Liashchynskyi, Petro and Pavlo Liashchynskyi (2019). "Grid search, random search, genetic algorithm: a big comparison for NAS". In: *arXiv preprint arXiv:1912.06059*.

Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2018). "Darts: Differentiable architecture search". In: *arXiv preprint arXiv:1806.09055*.

Lujan-Moreno, Gustavo A et al. (2018). "Design of experiments and response surface methodology to tune machine learning hyperparameters, with a random forest case-study". In: *Expert Systems with Applications* 109, pp. 195–205.

Luna, Jose et al. (2022). "Orthogonal array composite designs for drug combination experiments with applications for tuberculosis". In: *Statistics in medicine* 41.17, pp. 3380–3397.

McKay, M. D., R. J. Beckman, and W. J. Conover (1979). "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code". In: *Technometrics* 21.2, pp. 239–245. DOI: 10.2307/1268522.

McKay, Michael D (1992). "Latin hypercube sampling as a tool in uncertainty analysis of computer models". In: *Proceedings of the 24th conference on Winter simulation*, pp. 557–564.

Mezura-Montes, Efrñn, Jesús Velázquez-Reyes, and Carlos A Coello Coello (2006). "A comparative study of differential evolution variants for global optimization". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 485–492.

Michalewicz, Zbigniew (2013). *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media.

Miettinen, Kaisa (1999). *Nonlinear multiobjective optimization*. Vol. 12. Springer Science & Business Media.

Mitchell, Melanie (1998). *An introduction to genetic algorithms*. MIT press.

Mockus, J, V Tiesis, and A Zilinskas (1978). "The application of Bayesian methods for seeking the extremum". In: *Toward Global Optimization* 2, pp. 117–130.

Mockus, Jonas (1994). "Application of Bayesian approach to numerical methods of global and stochastic optimization". In: *Journal of Global Optimization* 4, pp. 347–365.

Molga, Marcin and Czesław Smutnicki (2005). "Test functions for optimization needs". In: *Test functions for optimization needs*.

Montgomery, Douglas C (2017). *Design and analysis of experiments*. John wiley & sons.

Morris, Max D and Toby J Mitchell (1995). "Exploratory designs for computational experiments". In: *Journal of statistical planning and inference* 43.3, pp. 381–402.

Pelikan, Martin, David E Goldberg, Erick Cantú-Paz, et al. (1999). "BOA: The Bayesian optimization algorithm". In: *Proceedings of the genetic and evolutionary computation conference GECCO-99*. Vol. 1. Citeseer, pp. 525–532.

Picheny, Victor, Tobias Wagner, and David Ginsbourger (2013). "A benchmark of kriging-based infill criteria for noisy optimization". In: *Structural and multidisciplinary optimization* 48, pp. 607–626.

Price, Kenneth, Rainer M Storn, and Jouni A Lampinen (2006). *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media.

Qin, A Kai, Vicky Ling Huang, and Ponnuthurai N Suganthan (2008). "Differential evolution algorithm with strategy adaptation for global numerical optimization". In: *IEEE transactions on Evolutionary Computation* 13.2, pp. 398–417.

Rasmussen, C and C Williams (2006). *Gaussian processes for machine learning*. MIT press: Cambridge, MA.

Regis, Rommel G (2016). "Trust regions in Kriging-based optimization with expected improvement". In: *Engineering optimization* 48.6, pp. 1037–1059.

Regis, Rommel G and Christine A Shoemaker (2007). "Improved strategies for radial basis function methods for global optimization". In: *Journal of Global Optimization* 37, pp. 113–135.

Roustant, Olivier, David Ginsbourger, and Yves Deville (2012). "DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization". In: *Journal of statistical software* 51, pp. 1–55.

Schwefel, Hans-Paul (1977). "Numerische optimierung von computer-modellen mittels der evolutionsstrategie". In: *(No Title)*.

— (1981). *Numerical optimization of computer models*. John Wiley & Sons, Inc.

Schwefel, Hans-Paul Paul (1993). *Evolution and optimum seeking: the sixth generation*. John Wiley & Sons, Inc.

Shi, Chenlu, Ashley Kathleen Chiu, and Hongquan Xu (2023). "Evaluating designs for hyperparameter tuning in deep neural networks". In: *The New England Journal of Statistics in Data Science* 1.3, pp. 334–341.

Shi, Yuhui and Russell Eberhart (1998). "A modified particle swarm optimizer". In: *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*. IEEE, pp. 69–73.

Snelson, Edward Lloyd (2008). *Flexible and efficient Gaussian process models for machine learning*. University of London, University College London (United Kingdom).

Snoek, Jasper, Hugo Larochelle, and Ryan P Adams (2012). "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems* 25.

Srinivas, Niranjan et al. (2009). "Gaussian process optimization in the bandit setting: No regret and experimental design". In: *arXiv preprint arXiv:0912.3995*.

Stokes, Zack, Weng Kee Wong, and Hongquan Xu (2024). "Metaheuristic Solutions to Order-of-Addition Design Problems". In: *Journal of Computational and Graphical Statistics* 33.3, pp. 1006–1016.

Storn, Rainer and Kenneth Price (1997). "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of global optimization* 11, pp. 341–359.

Sun, Fasheng, Yaping Wang, and Hongquan Xu (2019). "Uniform projection designs". In: *Annals of Statistics* 47.1, pp. 641–661.

Surjanovic, S. and D. Bingham (2013). *Virtual Library of Simulation Experiments: Test Functions and Datasets*. Retrieved July 12, 2024, from `http://www.sfu.ca/~ssurjano`. Simon Fraser University.

Tabachnick, B. G. and L. S. Fidell (2019). *Using Multivariate Statistics*. Pearson.

Talbi, EG (2009). "Metaheuristics: From Design to Implementation". In: *John Wiley & Sons google schola* 2, pp. 268–308.

Tang, Boxin (1993). "Orthogonal array-based Latin hypercubes". In: *Journal of the American Statistical Association* 88.424, pp. 1392–1397.

Vent, W (1975). *Rechenberg, Ingo, Evolutionsstrategie-Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. 170 S. mit 36 Abb. Frommann-Holzboog-Verlag. Stuttgart 1973. Broschiert*.

Welch, William J et al. (1992). "Screening, predicting, and computer experiments". In: *Technometrics* 34.1, pp. 15–25.

Whitley, Darrell (1994). "A genetic algorithm tutorial". In: *Statistics and computing* 4, pp. 65–85.

Wu, CF Jeff and Michael S Hamada (2011). *Experiments: planning, analysis, and optimization*. John Wiley & Sons.

Wu, Jia, SenPeng Chen, and XiYuan Liu (2020). "Efficient hyperparameter optimization through model-based reinforcement learning". In: *Neurocomputing* 409, pp. 381–393.

Xiao, Qian and Hongquan Xu (2018). "Construction of maximin distance designs via level permutation and expansion". In: *Statistica Sinica* 28.3, pp. 1395–1414.

Xu, Hongquan, Jessica Jaynes, and Xianting Ding (2014). "Combining two-level and three-level orthogonal arrays for factor screening and response surface exploration". In: *Statistica Sinica* 24.1, pp. 269–289.

Yang, Xin-She and Mehmet Karamanoglu (2013). "Swarm intelligence and bio-inspired computation: an overview". In: *Swarm intelligence and bio-inspired computation*, pp. 3–23.

Yang, XS (2010). *Engineering Optimization: An Introduction with Metaheuristic Applications*. John Wiley & Sons.

Zhou, Qing and Yanda Li (2003). "Directed variation in evolution strategies". In: *IEEE Transactions on Evolutionary Computation* 7.4, pp. 356–366.

Zoph, Barret and Quoc V Le (2016). "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578*.