TOWARDS UNDERSTANDING THE IMPACT OF PROBLEM LANDSCAPES IN NUMERICAL BLACK-BOX OPTIMIZATION

Urban Škvorc

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Prof. Dr. Peter Korošec, Jožef Stefan Institute, Ljubljana, Slovenia
Co-Supervisor: Asst. Prof. Dr. Tome Eftimov, Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:
Asst. Prof. Dr. Tea Tušar, Chair, Jožef Stefan Institute, Ljubljana, Slovenia
Dr. Carola Doerr, Member, CNRS and Sorbonne University, Paris, France
Asst. Prof. Dr. Hao Wang, Member, Leiden University, Leiden, Netherlands

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Urban Škvorc

TOWARDS UNDERSTANDING THE IMPACT OF PROBLEM LANDSCAPES IN NUMERICAL BLACK-BOX OPTIMIZATION

Doctoral Dissertation

RAZUMEVANJE VPLIVA PROBLEMSKIH POKRAJIN V NUMERIČNI OPTIMIZACIJI ČRNE ŠKATLE

Doktorska disertacija

Supervisor: Prof. Dr. Peter Korošec

Co-Supervisor: Asst. Prof. Dr. Tome Eftimov

Ljubljana, Slovenia, February 2023

To my family

Acknowledgments

This work would not have been possible without the constant support and guidance of my supervisors Prof. Dr. Peter Korošec and Asst. Prof. Dr. Tome Eftimov. I am deeply grateful for their support. Additionally, I would like to thank the evaluation board, consisting of Asst. Prof. Tea Tušar, Dr. Carola Doerr, and Asst. Prof. Hao Wang, for taking the time to evaluate this thesis.

I would also like to thank everyone at the department of Computer Systems at the Jožef Stefan Institute for creating a welcoming and supportive work environment, and always being open to providing advice. I would particularly like to thank Assoc. Prof. Barbara Koroušić Seljak, with whom I worked on several projects before the beginning of my Ph.D. studies, Asst. Prof. Anton Biasizzo for his mentorship during my Master's studies, and the secretarial team of Jolanda Jakofčič and Andreja Vlašič for always being ready to answer any of my questions.

I acknowledge the Slovenian Research Agency, which provided the funding for this thesis through the Young Researcher program under project number PR-08987 and through research core funding under project number P2-0098 .

I would like to also thank all of the anonymous reviewers whose feedback, both positive and negative, of the conference and journal papers related to this thesis helped me improve my research and writing skills. The work done by reviewers is often underappreciated, despite being a fundamental part of our current research process.

Finally, I would like to thank my family for their endless support during the course of my studies.

Abstract

In optimization, it is well known that algorithm performance is dependent on the problem being solved. As a consequence of this, achieving good optimization results requires correctly matching an optimization problem to a specific optimization algorithm that performs well on that problem. For this to be possible, knowledge of both optimization algorithms and optimization problems is required. However, while a large amount of work has been dedicated to analyzing and creating optimization algorithms, there is a relatively limited amount of research available that analyzes optimization problems. In this thesis, we focus on the analysis of numerical optimization problems using a popular state-of-the-art method called exploratory landscape analysis, which transforms samples collected from an optimization problem into numerical descriptors called landscape features.

First, we perform an analysis of the problem sets from two most popular single-objective numerical optimization benchmarks events. We show that the performance of optimization algorithms differs on the problems used at these events, which we assume is caused by the differences in the problems. We show that these differences are also present among the different years of a single benchmarking event, meaning that an algorithm that achieves the best performance on the most recent version of the event might not necessarily be the best-performing algorithm if problems from an older version of the same event are taken into account. Then, we further examine these two problem sets by calculating exploratory landscape analysis features on these problems. We particularly focus on examining the generalizability of exploratory landscape analysis, and show that some landscape features used for our experiments are highly sensitive to problem transformations of shifting and scaling.

Finally, we evaluate the suitability of exploratory landscape analysis for the task of selecting the best-performing optimization algorithm for a specific problem, called algorithm selection. Prior work has shown that it can be used for this task when the algorithm selection model is both trained and tested on the problems of a single benchmark set. In our experiments, we instead examine the generalizability of such a meta-learning method. We train an algorithm selection model on a set of algorithmically generated problems in order to create a training set that has never been examined before. We use this set to predict the best-performing algorithms on the problems of an existing optimization benchmark problem set. This experiment produces low-quality results, indicating that the algorithmically generated problems are not generalizable to the set of existing benchmark problems when exploratory landscape analysis is used to describe the problems.

The main contribution of the thesis is an in-depth and systematic examination of optimization problems, with a focus on exploratory landscape analysis. We show that while exploratory landscape analysis can be used to describe optimization problem landscapes, this must be done with caution and understanding that some landscape features can be sensitive to even simple problem transformations. Additionally, when used for algorithm selection, we show that these features might be unable to generalize among different sets of problems.

Povzetek

Na področju optimizacije velja, da je uspešnost optimizacijskih algoritmov odvisna od problemov, ki jih algoritmi rešujejo. Posledično moramo za dosego dobrih optimizacijskih rezultatov pravilno izbrati tisti algoritem, ki je zmožen ta problem dobro rešiti. To lahko storimo s podrobnim poznavanjem tako optimizacijskih algoritmov kot tudi problemov. Kljub temu pa je v trenutni znanstveni literaturi področje analize optimizacijskih problemov izrazito manj zastopano od področja analize algoritmov. V tej doktorski disertaciji se osredotočamo na analizo optimizacijskih problemov z uporabo metode raziskovalna analiza problemske pokrajine (angl. exploratory landscape analysis), ki vzorce problema pretvori v značilke problemske pokrajine.

V disertaciji najprej izvedemo analizo dveh najbolj znanih dogodkov za primerjavo uspešnosti optimizacijskih algoritmov. Pokažemo, da so rezultati merjenja uspešnosti na teh dogodkih različni, kar menimo da je posledica tega, da dogodki uporabljajo različne optimizacijske probleme. Poleg tega pokažemo, da so tovrstne razlike prisotne tudi med vsakoletnimi različicami problemov na enemu izmed teh dogodkov. To pomeni, da uspešnosti algoritmov, izmerjene na najnovejših različicah problemov, niso nujno primerljive s tistimi na prejšnjih različicah. Torej da algoritem, ki se izkaže za najboljšega na najnovejših različicah problemov, ni nujno tako uspešen, ko ga uporabimo za reševanje starejših različic problemov. Nato analiziramo pogosto uporabljeno metode raziskovalne analize preiskovanja problemske pokrajine za opis značilk optimizacijskih problemov. V sklopu te raziskave ugotovimo, da so nekatere značilke problemske pokrajine občutljive na preslikavi premika in središčnega raztega.

V zadnjem delu disertacije ovrednotimo uporabo značilk problemske pokrajine za napovedovanje najboljšega optimizacijskega algoritma za posamezen problem. Predhodne raziskave so pokazale, da se značilke obnesejo dobro, če se model uči na isti množici problemov, kot je uporabljena za napovedovanje. V naših eksperimentih preverimo, če to velja tudi, ko se model uči na povsem ločeni množici problemov. Za ta namen kot učno množico uporabimo nabor problemov ki so bili ustvarjeni z uporabo računalniškega algorithma, in jih uporabimo za napoved na naboru problemov iz obstoječe zbirke problemov. Rezultati, ki jih pridobimo, so negativni, kar lahko nakazuje, da množici podatkov, ki ju uporabimo za učenje in napovedovanje, nista dovolj splošni, ko probleme opisujemo z značilkami pokrajin.

Glavni prispevek disertacije je bolj podrobna analiza optimizacijskih problemov s poudarkom na analizi značilk pokrajine problema. V disertaciji pokažemo, da te značilke sicer lahko opišejo optimizacijski problem, vendar je pri njihovi uporabi potrebna previdnost in razumevanje, da so nekatere značilke občutljive na nekatere pogoste transformacije. Poleg tega pokažemo, da značilke v nekaterih primerih ne morejo posplošiti informacij med različnimi nabori problemov, ko jih uporabljamo za napovedovanje najboljšega optimizacijskega algoritma za posamezen problem.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ABC	... Artificial Bee Colony
ACO	... Ant Colony Optimization
aRT	... Average Runtime
BBOB	... Black-Box Optimization Benchmarking
BIPOP	... Bi-Population
CAL-SHADE	... Constrained Linear Population Size Reduction Success-History Based Adaptive Differential Evolution
CEC	... IEEE Congress on Evolutionary Computation
COCO	... Comparing Continuous Optimizers
CSO	... Competitive Swarm Optimizer
DE	... Differential Evolution
ECDF	... Empirical Cumulative Distribution Function
ELA	... Exploratory Landscape Analysis
ERT	... Expected Running Time
DSC	... Deep Statistical Comparison
FEP	... Fast Evolutionary Programming
flacco	... Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems
GECCO	... The Genetic and Evolutionary Computation Conference
HECO-DE	... Helper and Equivalent Objective Differential Evolution
I/F-Race	... Iterated F-Race
ICOP	... Interpolated Continuous Optimisation Problems
JADE	... Joint Approximation Diagonalization of Eigen-Matrices
IPOP	... Increasing Population Size
LHS	... Latin Hypercube Sampling
MATLAB	... Matrix Laboratory
NFL	... No Free Lunch
MOS	... Multiple Offspring Sampling
PCA	... Principal Component Analysis
PSO	... Particle Swarm Optimization
Rand	... Random Search
SA	... Simulated Annealing
SBS	... Single Best Solver
SEP	... Separable
SHAP	... Shapley Additive Explanations
SMAC	... Sequential Model-Based Optimization for General Algorithm Configuration
SPO	... Sequential Parameter Optimization
SVD	... Singular Value Decomposition
VBS	... Virtual Best Solver
VNS	... Variable Neighbourhood Search

Chapter 1

Introduction

Optimization, the task of finding a value of X that either maximizes or minimizes an objective function $f(X)$ of a particular optimization problem, is a field of mathematics with a large number of possible applications, from teaching a computer to play chess (Duro & de Oliveira, 2008) to optimizing the design of antennas (Zhu et al., 2007). Numerical optimization is a branch of optimization that optimizes functions for which the domain is limited to the set of real numbers $\mathbb{R}$. This is in contrast to for example discrete optimization, where the domain consists of a finite, discrete set.

There exists a large number of different optimization algorithms (Yang, 2014). However, it has been shown from both a practical and a theoretical perspective that optimization algorithm performance is heavily dependent on the specific problem being solved. Because of this, the knowledge of optimization problems is an important factor in achieving good optimization performance.

Despite this, a large amount of research on optimization focuses solely on creating new algorithms or improving existing algorithms to for example achieve good performance on benchmark problems, or to solve specific real-life optimization problems (i.e., most research is devoted to the *performance space* of optimization).

In this thesis, we present a more comprehensive overview of not just the performance space of optimization, but also of the relatively less studied task of the understanding of optimization problems (in other words, the *problem space* of optimization). Finally, we present an analysis of how the knowledge of these two spaces can be combined in order to select the best-performing algorithm on a specific optimization problem. In addition, we present the challenges and the limitations of the current state-of-the-art approaches to problem space analysis.

1.1 Motivation

In 1997, Wolpert and Macready published the paper titled *No free lunch theorems for optimization* (Wolpert & Macready, 1997). This paper introduced and proved two mathematical theorems called the No Free Lunch (NFL) theorems. The first of these theorems is shown in Equation 1.1, which is adapted from (McDermott, 2020). Here, F_{all} is the set of all possible functions, Perf is any performance metric, a_1 and a_2 are any two optimization algorithms, and d_m^y is the history of objective values y of the function f over a set of the first m steps of the algorithms a_1 and a_2 (McDermott, 2020).

$$\sum_{f \in F_{all}} \text{Perf}(d_m^y | f, m, a_1) = \sum_{f \in F_{all}} \text{Perf}(d_m^y | f, m, a_2) \tag{1.1}$$

In simple terms, this theorem states that for any two optimization algorithms, their average performance across all possible optimization problems will be the same, regardless of how their performance is measured. Schumacher et al. (2001) presents a stricter version of this theorem, which shows that this result holds when the averaging is performed over any subset of problems that is closed under permutation of objective values, not just across the set of all possible problems. A set of problems is closed under permutation of objective values when the problems in the set differ only by the permutation of a fixed set of problem values across the points in the search space.

However, the practical relevance of the theorems on optimization is limited, as they only apply when the performance of an algorithm is averaged over the entire set of problems or over the subsets of problems closed under permutation of objective values. In practice, optimization is usually concerned with only a limited subset of problems that are either interesting from a theoretical point of view or that have practical reasons for being solved. While the NFL theorems themselves do not hold over such a limited set of problems, the theorems still provide a theoretical foundation for an important fact of optimization: algorithm performance is influenced by the problem that the algorithm is solving.

Optimization algorithms performing differently depending on the specific problem being solved has also been shown to be true in practice even in conditions where the NFL theorems do not hold, i.e., when not considering the entire set of all possible problems or a set of problems closed under permutation of objective values. For example, when evaluating algorithms that attempted to solve the 2017 CEC Benchmarking competition problem set (Song et al., 2019), the best performing overall algorithm HECO-DE (Xu et al., 2019) achieved the best performance on only 11 out of the 28 benchmark problems. In the same competition, even the worst performing algorithm overall, CAL-SHADE (Zamuda, 2017), achieved the best performance on 5 of the 28 benchmark problems.

This clearly illustrates that even algorithms that might perform worse on average are still able to solve specific problems well, even when the problem set is small. Interpreting such results can be particularly problematic if some of the problems in the benchmark set are heavily correlated with one another. If the 11 problems solved well by HECO-DE are all correlated with one another, then we cannot say that it is the best-performing algorithm overall.

The fact that some benchmark problems are correlated with one another has been shown in practice. For example, Christie et al. (2018) show that benchmark functions of the widely used BBOB benchmark set (Hansen et al., 2009) are correlated from the perspective of the performance of the Differential Evolution (DE) algorithm , in the sense that there are several problems for which the DE algorithm produces similar results throughout its execution (i.e., most iterations of the optimization process produce correlated results).

These results call into question the practice of using small benchmark problem sets to determine which algorithms perform well, as it is entirely possible that if the problem set was extended with more problems that favor the CAL-SHADE, this algorithm would go from being the worst-performing to the best performing. In addition, it is not possible to generalize the results of a single benchmark problem set, as different benchmark sets might produce different results. For example, Y.-W. Zhang and Halgamuge (2019) show that there is a difference in algorithm performance when comparing three different sets of benchmark problems. Cenikj et al. (2022) does the same while comparing the performance of five different benchmark problem sets, and introduces a method that can create a more evenly distributed set of benchmark problems.

Because optimization algorithm performance is problem-specific, algorithm selection is a popular area of research in optimization. Algorithm selection, first introduced by Rice (1976), is the problem of finding a mapping between the problem and the algorithm

space so that each problem is mapped to the algorithm that best solves it. Kerschke and Trautmann (2019a) show that an automatic algorithm selection system can be used to improve optimization performance on a set of benchmark problems, reducing the expected runtime by half compared to using a single optimization algorithm on all problems.

We can see that when it comes to optimization, the ability to match an optimization problem to a specific algorithm is important. This task requires knowledge of not only the performance space of optimization algorithms but also of the problem space. However, the analysis of the problem space has received relatively little attention compared to the analysis of algorithm performance, i.e., the performance space.

The traditional method for describing and categorizing optimization problems is to use high-level descriptive features. For example, the Comparing Continuous Optimizers (COCO) benchmarking platform (Hansen et al., 2021) splits its noise-free black-box optimization benchmarking (BBOB) problem set (Hansen et al., 2009) into five categories: separable functions, functions with low or moderate conditioning, functions with high conditioning and unimodal, multi-modal functions with adequate global structure, and multi-modal functions with weak global structure. However, determining these high-level features requires expert knowledge of each individual problem.

In 2011, Mersmann et al. (2011) introduced Exploratory Landscape Analysis (ELA). This approach transforms eight high-level features identified by the authors into a number of low-level numerical feature classes that can be computed algorithmically using problem samples. The high-level features are global structure, multi-modality, separability, global-to-local optima contrast, search space homogeneity, plateaus, variable scaling, and basin size homogeneity. The main benefit of this approach is that low-level features can be computed algorithmically, without requiring expert knowledge of the specific problems. Furthermore, because the features are calculated from only the problem samples, they can be calculated for so-called black-box problems, for which the exact problem definition is unknown. Such problems are common in practice and might include for example complex simulation models. Since the original ELA paper (Mersmann et al., 2011), a large number of other low-level features have been proposed. Additionally, a software library called flacco (Kerschke & Trautmann, 2019b) was created to compute these low-level features.

Exploratory Landscape Analysis has proven to be effective in practice. For example, it was used by Kerschke and Trautmann (2019a) to create the previously mentioned algorithm selection model that reduced the estimated runtime by half.

While this landscape-aware algorithm selection approach approach has shown promising results, there have been relatively few theoretical studies into the generalizability and reliability of the low-level features. For example, most of the literature that analyzes exploratory landscape analysis is based on a single benchmark set of the 24 BBOB problems (Hansen et al., 2009), with little research done to show how well these features generalize between problem sets. One such research by Lacroix and McCall (2019) shows that while the landscape features perform well under the 24 BBOB problems in terms of predicting algorithm performance, this does not generalize to a different set of problems. Finally, Lang and Engelbrecht (2021) show that different benchmark problem sets have different coverage of the problem space, and propose a new set of benchmark problems with broader problem coverage.

Additionally, a recent article by Renau et al. (2020) has shown that these features are highly sensitive to the sampling strategy used to draw the samples. These results illustrate that there is currently a lack of research that focuses on an in-depth understanding of the problem landscape of optimization, and how this landscape relates to the performance landscape of optimization.

In this thesis, we aim to improve the current state of the art on this topic. We will do so

by examining numerical optimization from the perspectives of both the performance and the problem space. Finally, we will attempt to combine both of these spaces by performing algorithm selection in order to automatically determine which optimization algorithms achieve the best performance on which optimization problem.

1.2 Hypotheses

This thesis investigates the following three hypotheses related to the analysis of numerical optimization problems.

Hypothesis 1 The development of novel optimization algorithms in recent years does not lead to statistically significant improvement in performance indicating that the research is becoming ever more incremental.

Hypothesis 2 The problem space of optimization as described by exploratory landscape analysis features can be sensitive to the transformations of shifting and scaling.

Hypothesis 3 Exploratory landscape analysis does not represent optimization problems generally enough to use it for automated algorithm selection if the training and testing sets contain problems with large differences in design. Specifically, if the algorithm selection model is trained on an algorithmically generated set of problems and evaluated on a benchmark set designed by domain experts.

1.3 Contributions

The work presented in this thesis has made the following scientific contributions.

1. We show that algorithms that have achieved the best performance in a particular year of an optimization benchmarking competition do not necessarily outperform algorithms from other years. This has several implications. First, it shows that when evaluating the performance of an optimization algorithm, it is important to not only consider algorithms that have won the most recent year of the competition, as older algorithms might outperform newer algorithms on certain problems. Second, it shows that current benchmarking events are not a good way of measuring overall algorithm performance.

2. We show that some exploratory landscape features are not invariant to the transformations of shifting and scaling. These transforms are commonly used by optimization benchmarks to create new problem instances because they do not impact the performance of state-of-the-art optimization algorithms. For this reason, the information provided by the non-invariant landscape features might not correlate with algorithm performance.

3. We create and test a machine learning model for algorithm selection, and show that the model achieves good performance when trained and tested on problems from a single benchmark set. However, when the model is used for more generalized learning, i.e., when it is trained on one set of problems and tested on a different set of problems, it achieves a much lower performance. This shows that to obtain good results, it is currently important to use similar problems in both the training and the testing set, and further shows the difficulty of obtaining generalized information using exploratory landscape features across different problem sets, that is, information that is not specific to a single problem set, but that can be used generally and independently of the problem set.

1.4 Thesis Structure

The rest of the thesis is structured in the following way. In Chapter 2, we present an overview of existing research necessary for the understanding of this thesis. In this chapter, we describe numerical optimization benchmarking and the two benchmark sets that will be examined in this thesis, as well as the methods for analyzing optimization problems. We then provide a brief overview of methods used to describe optimization problems, including exploratory landscape analysis. We conclude this chapter with an overview of techniques used for algorithm parameter tuning and for algorithm selection.

In Chapter 3, we examine the performance space of optimization by evaluating the results of algorithms from two popular benchmarking events. We compare the performance of algorithms submitted in the different years of the benchmark events to determine whether newer algorithms achieve performance that is statistically significantly better than the algorithms from the previous years. If this is not the case, it indicates that newer algorithms are tuned to the problems from the specific benchmark year, but do not produce overall improvements over a broader set of problems.

In Chapter 4, we conduct a thorough examination of the problem space of optimization. In particular, we examine the reliability of exploratory landscape analysis, a commonly used method for describing optimization problems by using numerical values. We verify the results of our reliability experiments by using exploratory landscape analysis to investigate the complementarity of optimization problems belonging to different benchmark sets.

In Chapter 5, we combine the knowledge of both the performance space and the problem space by attempting to select the best-performing algorithm for a specific optimization problem, also called automated algorithm selection. We attempt to predict the best-performing algorithm on a set of existing optimization benchmark problems, using information obtained from a separate set of algorithmically generated problems. This scenario is designed to examine whether the current state-of-the-art problem space analysis methods can be used to obtain generalized information that is not specific to a single type of optimization problem.

In Chapter 6, we present the summary and conclusions of our work, describe which of our initial hypotheses were confirmed by the work in this thesis, and describe plans for future work.

All in all, this thesis represents an in-depth examination of both the performance and problem spaces of optimization, and of how they can be combined for the practical application of algorithm selection. This research covers a crucial research gap that exists in current literature, particularly in terms of the reliability and the generalizability of current state-of-the-art problem space analysis methods.

Chapter 2

Background

This chapter contains an overview of existing background work that is necessary for the understanding of this thesis. In the first part of this chapter, we introduce the performance space of numerical optimization. We describe the basics of optimization, as well as the performance measures and techniques used to compare optimization algorithms. Then, we describe the two popular numerical optimization benchmarking events that will be examined in our experiments: the CEC Special Sessions & Competitions on Real-Parameter Single Objective Optimization and the Black-Box Optimization Benchmarking Workshops.

In the second part, we describe the problem space of optimization. We present several techniques that can be used to analyze optimization problems, including both high-level and low-level techniques.

In the third part, we present several ways of combining the knowledge of both the performance and the problem spaces. We begin by describing the tasks of parameter tuning and parameter control, which deal with selecting the correct set of algorithm parameters for a specific problem being solved. Finally, we describe the algorithm selection problem, i.e., the problem of automatically selecting the best performing algorithm for a specific optimization problem.

2.1 Numerical Black-Box Optimization Benchmarking

In this section, we present an overview of the performance space of optimization. The performance of optimization algorithms is often measured by using optimization benchmarks, i.e., sets of predefined optimization problems and evaluation metrics.

We first give a mathematical definition of numerical optimization. We then provide a general description of the statistical methods that can be used to compare optimization algorithms with one another. Finally, we describe two popular numerical optimization benchmark sets and their evaluation metrics.

2.1.1 Optimization

From (Bonnans et al., 2006) given X and a function $f : X \to \mathbb{R}$, the process of global optimization is the process of finding such $x^* \in X$ that for all $x \in X$, $f(x^*) \leq f(x)$. such a point is also called the global optimum. In addition to the global optimum, a function can also contain local optima, which are points x^*_{local} for which $f(x^*_{local}) \leq f(x_{local})$ for all $(x_{local} \in [x^*_{local} - \epsilon, x^*_{local} + \epsilon]$ for some ϵ value. The area of $[x^*_{local} - \epsilon, x^*_{local} + \epsilon]$ is also called the $\epsilon - neighbourhood$

Additionally, it is common for optimization problems to contain constraints that limit the values of X to a subset of $\mathbb{R}^D$. In mathematical terms, the problem of numerical

optimization can be stated using the set of equations:

$$\begin{aligned} &\min f(x) x \in \mathbb{R}^D \\ &g_i(x) \leq 0 j = 1, ..., n_g \\ &h_j(x) = 0 i = 1, ..., n_h \end{aligned} \tag{2.1}$$

Where g_i are equations that define the inequality constraints of x and h_j are equations that define the equality constraints of x. In this thesis, we will focus on unconstrained optimization (where the set of constraint equations is empty and x is allowed to take any form) and on bound constrained optimization (where the only constraints are that $x_{\min} \leq x \leq x_{\max}$). Boundary constraints are commonly used in numerical optimization, as without them the search space of an optimization problem would be infinitely large. Such constraints are also naturally present in real-life optimization problems, where values of x represent some real-life parameters that cannot be infinitely large.

In this dissertation, we will focus primarily on black-box optimization, for which the objective function f is unknown, unexploitable, or non-existent (Alarie et al., 2021).

Optimization problems are solved by optimization algorithms. One of the simplest examples of such an algorithm is the gradient descent algorithm (Netrapalli, 2019). Gradient descent works by starting at a random point of an optimization problem and moves in the direction of the gradient (or a numerical approximation when the exact gradient cannot be computed) at every iteration. This ensures that a local optimum will be found, but the algorithm will only be able to find a single local optimum from its starting point.

Because searching the entire problem space is a time-consuming task, optimization algorithms are commonly designed to find an approximate rather than the absolute best solution. In addition, most algorithms are also stochastic, meaning that they involve randomness in their execution.

One popular family of optimization algorithms that we will examine as a part of this thesis are population-based algorithms. An example of the basic structure of these algorithms is shown in Algorithm 2.1. Population-based algorithms operate on a set of individual problem samples called individuals that together form a single population. At the beginning of the algorithm, a population is generated using some procedure, for example, random sampling. Then, the individuals are evaluated to obtain the value of the objective function for each of the individuals.

After these initial steps, the algorithm enters into a loop. In every iteration of the loop, the current population is used to create a new set of individuals. After this new set is generated, a selection procedure is used to select which of the new and existing solutions are kept in the population, and which are discarded. This loop continues until some stopping criteria is met. The loop can for example be terminated after a specified amount of time or resources has been used, if the algorithm has executed a specific number of iterations without improving on the best-found result, or if a solution has been found that is better than some target value.

This family of algorithms differs from a traditional gradient descent (Netrapalli, 2019) because it uses information from multiple problem samples in every iteration, rather than just a single sample used in gradient descent. In addition to the category of algorithms shown in Algorithm 2.1, other categories of optimization algorithms exist. One such example are surrogate optimization algorithms, which use an estimate of the optimization problem being solved to reduce computational time. Stork et al. (2022) present a more detailed taxonomy of optimization algorithms in .

Algorithm 2.1: A generic population-based optimization algorithm.

$n \leftarrow population\_size$;
$d \leftarrow problem\_dimension$;
$population \leftarrow generate\_population(n, d)$;
while *stopping criteria not met* **do**
  $results \leftarrow f(population)$;
  $population\_new \leftarrow transform(population)$;
  $results\_new \leftarrow f(population\_new)$;
  $population \leftarrow selection(population, population\_new, results, results\_new)$;
end

2.1.2 Performance Measures

One difficulty of evaluating the performance of optimization algorithms is that there exists no single commonly agreed upon way of doing so. Instead, multiple performance measures are used in literature with the goal of describing different aspects of an algorithm's performance.

The simplest approach to analyzing optimization algorithm performance would be to measure the time, specified in the number of function evaluations, that an algorithm needs in order to find an optimal solution to the problem (Jansen, 2013), or some other solution that is considered sufficiently close to the optimum. We note that when evaluating the execution times of optimization algorithms, most performance metrics are primarily concerned with the number of *function evaluations* that the optimization algorithm performs, where a function evaluation is a single calculation of the problem $f(x)$ that is being optimized. The remaining computational complexity of the algorithm is often either completely ignored or evaluated in a separate experiment. The justification of this approach is that when solving real-life problems, the time to evaluate the function $f(x)$ is often orders of magnitude longer than any other part of the algorithm.

Current performance measures of optimization algorithms primarily use one of two approaches: *fixed target* (Buzdalov et al., 2020) or *fixed budget* (Jansen & Zarges, 2014). In the fixed target approach, we measure how fast an algorithm can achieve a result that is at least as good as some fixed target. However, this requires us to first specify the target. In some cases, for example in benchmarking, this target can simply be the optimum of the problem being optimized, or more often, any solution that is within some fixed margin of error from the exact optimum. However, in some cases, such as in real-life problems, the optimum of a problem might not be known in advance. In this situation, some other target has to be chosen. For real life problems, this could be the worst solution that is still considered acceptable to the people solving the problem.

In the fixed budget approach, we measure the best result obtained by an algorithm after a specified number of function evaluations. Unlike the fixed target approach, the fixed budget approach does not require us to specify a target in advance. However, it also does not guarantee that the optimization algorithm will find any acceptable solutions using the specified budget, and provides no information on performance past the fixed budget.

Many optimization algorithms are stochastic in nature, meaning that they do not necessarily produce the same result every time they are executed. Because of this, the distribution of the results produced from multiple runs of the algorithm, or a single value such as the mean or the median of the runs, can be used.

More advanced performance measures exist which take into account the stochastic nature of optimization algorithms. One such example is the metric called average runtime

(aRT) (Auger & Hansen, 2005; K. V. Price, 1997) used by the Black-Box Optimization Benchmarking Workshops, which measures the expected number of function evaluations needed to obtain a result better or equal to the target value for the first time. This metric will be explained in more detail in Section 2.1.4.2, which describes the Black-Box Optimization Benchmarking Workshops.

2.1.3 Comparisons Between Algorithms

Due to the stochastic nature of optimization algorithms, it is standard practice to compare the performance of multiple different optimization algorithms by using statistical tests to determine if the differences between algorithms are statistically significant.

Two different kinds of statistical tests exist: parametric and nonparametric. Generally speaking, parametric tests have more statistical power, but require the data they are used on to satisfy specific conditions. These conditions are independence (the data samples are independent from one another), normality (the samples follow a normal distribution), and homoscedasticity (all of the sets of data being compared have the same variance) (Sheskin, 2003).

Nonparametric tests are generally less powerful, however they do not require the data to satisfy the strict conditions that are required for parametric tests. Because data obtained from optimization algorithms in most cases does not fulfill the criteria for the use of parametric tests (García et al., 2009), nonparametric tests are commonly used in the field of optimization.

The comparison of algorithms can be carried out in two different ways. The first, pairwise comparison, tests for a difference between two algorithms. For these types of comparisons, the Wilcoxon signed-rank test, the nonparametric alternative to Student's t-test, is commonly used. The second, multiple algorithm comparison, compares more than two algorithms with one another. An example of a multiple comparison test is the Friedman test. When comparing multiple algorithms, the p-value might need to be adjusted using techniques such as the Bonferroni correction to avoid the multiple comparison problem.

One drawback of statistical testing commonly used in the field of optimization is that it only uses a single value, for example the mean or the median of the results, to compare algorithms. This can present problems when outliers are present in the data, as they can have a large effect on the mean of the data (Eftimov et al., 2016). Even if the median is used instead of the mean, a problem can arise in the case when the medians are within a certain small ϵ-neighborhood. While such small differences are not meaningful in practice, some tests such as the Friedman test do not account for the magnitude of the difference when performing algorithm ranking, meaning that such a test would consider samples with such a small difference in medians as different in terms of performance.

Eftimov, Korošec, and Koroušić Seljak (2017) present a method to address this issue, called Deep Statistical Comparison (DSC) . This method uses a two-sample Kolmogorov–Smirnov or the Anderson-Darling test to determine if any of the algorithms produced results that have the same distribution. This allows it to detect algorithms with similar performance that would be treated as different by common statistical approaches, or to differentiate between algorithms with similar medians, but with different distributions.

2.1.4 Numerical Black-Box Optimization Benchmarks

In this subsection, we present two popular numerical optimization benchmarking events. We first present the CEC Special Sessions & Competitions on Real-Parameter Single Objective Optimization, a long running optimization competition that varies its problem set between the years of the competition. We then present the Black-Box Optimization

Benchmarking Workshops, a workshop series that uses the popular COCO benchmarking platform. This workshop series uses a constant set of benchmark problems with only small changes throughout the years. We present both the problems used by the benchmarks, as well as their evaluation methods.

2.1.4.1 CEC Special Sessions & Competitions on Real-Parameter Single Objective Optimization

The CEC Special Sessions & Competitions on Real-Parameter Single Objective Optimization (CEC Competitions) are numerical benchmarking competitions held at the IEEE Congress on Evolutionary Computation (CEC) annually since 2005. From 2005 to 2012, the competitions changed their focus every year. Since 2013, the competitions have focused on numerical single objective optimization. In this thesis, we will focus on the competitions that took place from 2013 onward, as these competitions all focus on single objective optimization, and use a similar implementation for all of their problems and algorithms. Throughout the years, the CEC competitions have used different problem sets depending on the year of the competition. In this thesis, we will use four problem sets from the competitions that took place in 2013 (J. J. Liang, Qu, Suganthan, & Hernández-Díaz, 2013), 2014 (J. J. Liang, Qu, & Suganthan, 2013), 2015 (J. J. Liang et al., 2014), and 2017 (Awad, N. H. and Ali, M. Z. and Liang, J. J. and Qu, B. Y. and Suganthan, P. N., 2016). In these problem sets, the CEC competitions group the problems into three categories:

Simple problems contain traditional well known theoretical benchmark problems, such as the Rastrigin (Rastrigin, 1974) and the Rosenbrock (Rosenbrock, 1960) functions.

Hybrid problems combine multiple simple problems by splitting the value of x into several sub-components, and assigning each sub-component to a separate problem.

Composition problems similarly combine both simple and hybrid problems using a weighted sum. However, in composition problems, the entire set of variables of x is used for every problem that makes up the composition problem.

In addition to the three groups above, every individual problem is assigned a number of high-level features that describe the properties of the optimization problem. Some of these features are modality, separability, rotation, conditioning, and differentiability.

The algorithms taking part in the competitions are evaluated using a fixed-budget scenario using multiple budgets, with exact budgets used varying throughout the different competitions. In addition, the algorithms are required to report the real-time runtime of the algorithm using a procedure that accounts for the differences in the hardware used to run the algorithms, with the exact procedure varying throughout the years.

2.1.4.2 Black-Box Optimization Benchmarking Workshops

The second of the two benchmark events are the single-objective noiseless functions used by the Black-Box Optimization Benchmarking Workshops. The workshops have been held mostly annually as part of the Genetic and Evolutionary Computation Conference since 2009. In total, 11 workshops have been held in the years 2009, 2010, 2012-2019, 2021, and 2022 as well as an additional session in 2015 held at the IEEE Congress on Evolutionary Computation.

The workshop series is run using the Comparing Continuous Optimisers (COCO) benchmarking platform, which provides several sets of benchmark problems, as well as a way to evaluate algorithms running these problems. Unlike the CEC competitions, the

BBOB workshops do not vary the problems used each year. The workshops evaluate the performance of algorithms on a number of different problem sets. The platform includes for example, a noiseless, a noisy, and a bi-objective problem set, among others. In this thesis, we will examine only the noiseless single-objective problem set (Hansen et al., 2009), which contains 24 benchmark problems. We will refer to these problems as the BBOB problems. For each of these problems, we can create an arbitrary number of problem variations called problem instances. These instances are created by performing various transformations on the original problem, for example scaling or shifting the base problem. To differentiate between the problems and their instances, we will refer to the 24 problems as the base BBOB problems. The 24 problems are divided into five groups, analogous to the high-level descriptors of the CEC problems. These groups are: separable problems, problems with low or moderate conditioning, problems with high conditioning and unimodal, multi-modal problems with adequate global structure, and multi-modal problems with weak global structure.

The BBOB workshops evaluate the performance of algorithms using a metric called average runtime (aRT) (Auger & Hansen, 2005; K. V. Price, 1997), sometimes also referred to as expected running time (ERT) (Hansen et al., 2012).

$$aRT = \frac{\#FE(f_{\text{best}} \geq f_{\text{target}})}{\#succ} \tag{2.2}$$

Where $\#FE(f_{\text{best}} \geq f_{\text{target}})$ is the total number of function evaluations where the best achieved result of the algorithm was equal to or greater than a specified target value, and $\#succ$ is the number of successful runs of the algorithm, i.e., runs in which the algorithm found a solution that is lower or equal to f_{target}. This metric assumes that an algorithm being evaluated will be executed multiple times, which accounts for the stochastic nature of optimization algorithms.

In addition to the aRT metric, the BBOB workshops present results using empirical cumulative distribution functions. This approach allows for measuring the performance of algorithms using multiple different targets, while also allowing for the examination of the performance at any number of function evaluations by evaluating the ratio of the selected targets that were solved at that number of evaluations (Hansen et al., 2022). Figure 2.1 shows an example of an empirical cumulative distribution function (ECDF) graph that is used to represent these results. The horizontal axis represents the number of function evaluations. The vertical axis represents the proportion of function, target pairs that have been successfully solved under the given number of function evaluations. Each function, target pair is successfully solved if the algorithm achieved a result at least as good as a specified target value on a specified problem instance. In this specific case, the function, target pairs are defined by using 15 instances of the sphere problem and 51 different targets ranging from 100 to 10^{-8}. In future chapters, we will refer to this type of comparison methodology as the *ECDF approach*. We can see that as the number of evaluations increases, so does the proportion of solved function, target pairs.

Finally, the BBOB workshops evaluate their results using the Wilcoxon rank-sum statistical test that compares the aRT ratios of each algorithm on each target. Hansen et al. (2022) provides a more detailed description of the performance metrics used in the BBOB benchmarks.

2.2 Optimization Problem Analysis

In order to analyze and understand optimization problems, we first need some way to describe their properties. In this section, we present different ways that can be used to

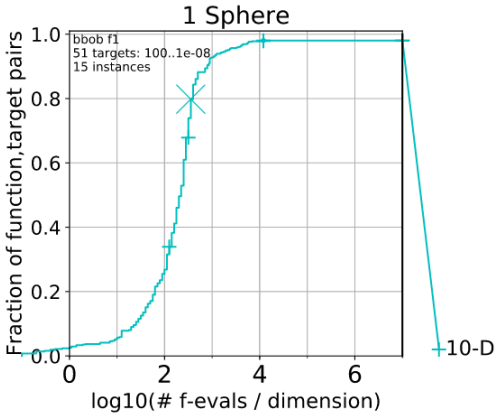

Figure 2.1: An example empirical cumulative distribution function used to present results of the BBOB workshops. The horizontal axis represents the number of function evaluations, while the vertical axis represents the proportion of solved function, target pairs.

describe optimization problems. We begin with the overview of high-level features, which use natural language to describe the properties of an optimization problem. However, high-level features are only descriptive, and need to be determined manually using expert knowledge. We continue with a presentation of low-level features that can be computed using a method called exploratory landscape analysis (Mersmann et al., 2011). These features do not require expert knowledge, as they can be computed automatically using mathematical formulas or computer algorithms. However, low-level features might not be as intuitive to humans as high-level features.

2.2.1 High-Level Features

A traditional way of describing optimization problems is using high-level features. These are natural-language descriptors that describe some property of an optimization problem, for example its modality or whether or not it contains plateaus.

The high level features provide an easy to understand, human language method for describing optimization problems. However, these features require a manual examination of the problem, and might require knowledge of an entire optimization problem. For example, knowing the modality of a problem requires the knowledge of all optima of a problem. Such knowledge is available for well-studied benchmark problems, but might be hard or impossible to obtain for other kinds of problems.

In this subsection, we list some of these high-level descriptors. First, we describe the properties that are used by the BBOB workshops described in the previous section (Hansen et al., 2009). We then describe an additional set of descriptors presented by Mersmann et al. (2010).

Examples of the high-level features used by the BBOB workshops and the CEC competitions are:

Separability describes whether or not a problem can be split into multiple separate problems of a lower dimension. Algorithms can simplify solving separable problems by solving each of the lower dimensional problems individually.

Modality describes the amount of local optima. Uni-modal problems only have a single local optimum (which is also the global optimum), while multi-modal problems con-

tain multiple local optima. These local optima can present problems for optimization algorithms, as they might get "trapped" in a local optimum.

Conditioning describes how small changes in the values of x influence the value of y by using a value called the condition number. A larger condition number results in higher changes of y. Problems with a high condition number are said to be ill-conditioned, while those with a small number are said to be well-conditioned.

Global structure describes the rough "shape" of an optimization problem. For example, in a single funnel structure, any decrease in results indicates that we are on average getting closer to the global optimum (Lunacek et al., 2008). Such a structure can be exploited by optimization algorithms.

Differentiability describes whether the problem is fully differentiable, only differentiable in certain regions, or completely non-differentiable. Some optimization approaches can only work on differentiable problems.

Mersmann et al. (2010) present several additional high-level descriptors that can be used to describe the BBOB problems. These are:

Variable scaling is similar to conditioning in that it measures the changes of result y in relation to the changes of the x. However, variable scaling measures how much the individual variables of x affect the result y. For example, some variables might produce large changes of y with even minor changes, while other variables might require large changes to influence y.

Search space homogeneity measures the difference in search space between different points in the landscape of the problem. In other words, do different parts of the problem all appear similar to one another, or does moving through the problem greatly change its landscape?

Basin size homogeneity measures the differences between the sizes of basins of attraction. A basin of attraction $S(x^*_{local})$ is defined as the area around a local optimum x^*_{local} for which running a strictly descending local optimizer with an infinitely small step size will always find x^*_{local} if it starts from any point $x \in S(x^*_{local})$.(Törn et al., 1999).

Global to local optima contrast measures the difference in problem value y between the local and the global optimum compared to the average of the entire problem. A global optimum that is greatly different from the local optima will be easier to find than a global optimum that is very similar to the local optima.

Plateaus are areas of a problem where the gradient is equal to zero. Such areas present problems for optimization algorithms, as they provide no directional information to guide the optimization algorithm.

2.2.2 Exploratory Landscape Analysis

In 2011, extended their previous work on high-level problem descriptors (Mersmann et al., 2010) by introducing exploratory landscape analysis (Mersmann et al., 2011). With ELA, the authors present six new categories of low-level descriptors that can be used to describe optimization problems by performing mathematical computations on a set of samples from a given problem. Because of this, the low-level features can be computed algorithmically without expert knowledge. Further, because these features are computed using problem

samples, they do not require knowledge of the entire problem or its exact definition. However, as the features are based on samples, they only represent an approximation of the underlying problem, and can be sensitive to the choice of sampling size and strategy (Renau et al., 2020).

Each of the six categories of low level features was designed to represent a number of high-level features that were presented in the previous section. The low-level feature categories introduced by the authors are:

Convexity selects two random samples, and a new point is selected as a convex combination of the original point. The value of the new point is then compared to the convex combination of the value of the two original points. This procedure is then repeated multiple times. An additional function evaluation is required for each repetition to calculate the value of the new point. This low-level feature category maps to the high-level features **global structure**, **search-space homogeneity**, **basin size homogeneity**, and **modality**.

y-Distribution describes the distribution of problem values y. This is measured in a number of different ways, for example by calculating their skewness ($\frac{\mu_3}{\sigma^3}$, where μ_3 is the third central moment of the function, and σ is its standard deviation), kurtosis ($\frac{\mu_4}{\sigma^4}$), and by estimating the number of peaks. This low-level feature category maps to the high-level features **global structure**, **modality**, and **plateaus**.

Levelset features split the original samples into two sets based on whether the problem value of the samples is higher or lower than a given threshold value (which can for example be the median of all values). A prediction model then predicts to which each sample belongs, and metrics of this model (for example the mean misclassification error) are used as features. This low-level feature category maps to the high-level features **modality** and **search-space homogeneity**.

Meta-Model features are computed by fitting either a linear or a quadratic model to the samples. The metrics of these fits, for example the model fit, linear intercept, or the condition of the quadratic model are used as features. This low-level feature category maps to the high-level features **global structure**, **separability**, **plateaus**, and **variable scaling**.

Local search features are based on a run of a local search optimizer that starts from a random sample. The metrics and results of this local search are then used to calculate the landscape features in this category. This low-level feature category maps to the high-level features **modality**, **global to local optima contrast**, **search-space homogeneity**, and **basin size homogeneity**.

Curvature features are based on the estimates of the gradient and the Hessian matrix (matrix of second order partial derivatives) of a random subset of samples. This low-level feature category maps to the high-level features **search-space homogeneity**, **plateaus**, and **variable scaling**.

Since the publication of the original ELA paper, a number of additional low-level features have been proposed in literature, and an open source library flacco (Kerschke & Trautmann, 2019b), which is available in both Python and R, has been created with the goal of collecting as many new and existing features as possible. These additional features do not necessarily map clearly to the high-level features. The flacco library contains the following additional low-level feature categories:

Cell mapping features discretize the problem space into a number of evenly sized cells. The cell mapping features are then computed using the information from each cell, for example the angle between the most central sample of the cell and the samples with the highest and lowest problem value (Kerschke et al., 2014) .

Generalized cell mapping features likewise discretize the problem space into cells, but these features pick only a single sample of each cell as that cell's representative (for example the sample with the highest or the lowest value), and then model the relations between the cells (Kerschke et al., 2014).

Barrier Tree features are similar to the generalized cell mapping features, as they also model the relations between the points in the search space. Instead of discretizing the problem space into cells, these features represent the problem space as a barrier tree, with leaves representing the local optima (Flamm et al., 2002).

Nearest better clustering features measure the difference between the distances of a sample to its nearest neighbor and its nearest better neighbor (Kerschke et al., 2015).

Dispersion features compare the dispersion between all of the samples and a subset of the samples selected according to their problem value (Lunacek & Whitley, 2006).

Information content features are based on the concept of information content, which describes smooth a particular problem is (Muñoz et al., 2014).

Further miscellaneous features are a collection of simple features which do not fit into any other category. These are split into three subcategories. **Basic** features contain basic information, for example the number of samples and their bounds. **Linear model** features are based on a linear model fitted onto each of the cells used for the cell-mapping features. The final subcategory of features is based on **principal component analysis** (Wold et al., 1987) of the samples, with features representing for example how many components are required to explain 90% of the variance.

A more detailed description of these features is presented in (Kerschke & Trautmann, 2019b). Since the ELA features provide numerical descriptors of the optimization problems, they can be used to visualize data using for example principal component analysis. Some examples of this are presented in the work done by Muñoz and Smith-Miles (2017), Lacroix et al. (2017), and Lacroix and McCall (2019). Their work also shows another use case of ELA: the generation of artificial problems to complement existing benchmark problems by finding areas of the problem space that are not currently covered by existing problems, and generate new ones that fill this empty space. An example of this has also been shown by Muñoz and Smith-Miles (2020).

2.3 Parameter Tuning, Control, and Algorithm Selection

As we have previously described, the performance of optimization algorithms is dependent on the problem that they are solving. This extends not just to a specific algorithm, but also to the specific parameters that the algorithm is using. Because of this, the tasks of parameter tuning, parameter control, and algorithm selection are very popular research topics.

We begin with an overview of parameter tuning, which is used to select the best performing algorithm parameters for a specific algorithm, usually by executing the algorithm multiple times and using the information obtained from the multiple executions to select

the best possible parameters. We then describe parameter control, which is similar to parameter tuning, but allows the algorithm to change its parameters during execution. This has the potential to improve algorithm performance even further, as it allows the algorithm to adapt its parameters not just to the optimization problem, but rather to different regions of the optimization problem which might require different parameters. Finally, we describe algorithm selection, the task of choosing the best performing algorithm for a specific optimization problem.

2.3.1 Parameter Tuning and Control

Unlike algorithm selection, which is concerned with selecting the best algorithm for a given problem, algorithm parameter tuning selects the best parameters of an individual algorithm for the given problem. This is an important task, because just as different problems are solved well by different algorithms, so do different algorithm parameters work well for different problems.

In this subsection, we describe the work done on the related tasks of parameter tuning and parameter control. Parameter tuning is the task of selecting the best algorithm parameters for a given problem before the execution of an algorithm, while parameter control is the task of updating the algorithm's parameters during execution (Eiben et al., 1999).

Parameter tuning and control can be performed manually, for example by an expert deciding on the best possible set of parameters, or manually evaluating algorithm performance using different parameters, and choosing the best set of parameters. Manual parameter control can be performed by deciding on a set of rules the algorithm should follow. One such example is reducing the population size during algorithm execution, which shifts the focus of the algorithm from exploration to exploitation. In this subsection, we will instead focus on automated tuning and control, that is, methods where the parameters are set automatically by an algorithm, rather than by hand.

2.3.1.1 Parameter Tuning

Huang et al. (2019) present a survey of state-of-the-art optimization algorithm parameter tuning approaches. They categorize existing approaches as belonging to one of three categories:

Simple generate-evaluate methods first generate multiple sets of algorithm parameters. These parameters are then evaluated, and the best set is chosen. The methods belonging to this category differ based on how the sets of parameters are generated, and in the method used to evaluate the parameters. The simplest example of such a method is a brute force approach that simply runs an optimization algorithm on a large number of parameter sets and selects the best performing one. However, such an approach requires a large amount of computational resources, since the optimization algorithm has to be executed many times. A less computationally expensive approach is the F-Race algorithm (Birattari et al., 2002), which evaluates algorithms at multiple points during their execution, and immediately eliminates those that show statistically poor performance as determined by a Friedman statistical test. Using this procedure, poorly performing parameter sets are removed early, so that computational power can instead be used to evaluate more promising sets.

Iterative generate-evaluate methods unlike simple generate-evaluate methods that only generate and evaluate parameter sets once, iterative generate-evaluate methods perform this step multiple times in a loop, using the parameter sets that performed

well in the current iteration as a base to generate parameter sets for the next iteration. These methods can start with a much smaller set of parameters, as they do not need to search the entire parameter space in a single iteration. This is currently the most popular method of automated parameter tuning and can be further split into four subcategories.

Experimental design-based tuning approaches use experimental design techniques to tune algorithm parameters. One example of such an approach is CALIBRA (Adenso-Diaz & Laguna, 2006).

Numerical optimization-based methods use derivative-free optimization methods for sampling the possible parameter settings. One such example is presented by Yuan et al. (2012). The drawback of such methods is that they are generally limited to only tuning numerical parameters.

Heuristic search-based methods generate parameter samples by using some heuristic rules, such as ones used by optimization algorithms. One example of such a method is the Iterated F-Race (I/F-Race) (Balaprakash et al., 2007). I/F-Race works similarly to the traditional F-Race algorithm. However, instead of only performing a single iteration, I/F-Race performs the racing procedure multiple times, with the best parameters from a previous iteration used to create the parameters for the new iteration. Another popular method from this category is ParamILS (Hutter et al., 2009), which uses an iterated local search to select the parameters to be evaluated.

Model-based optimization approaches design a surrogate model that describes the performance of an optimization algorithm based on its parameters but without the computational complexity of evaluating the original problem. This model is then used to determine which parameters to evaluate. Two examples of such methods are Sequential Parameter Optimization (SPO) (Bartz-Beielstein et al., 2005), and Sequential model-based optimization for general algorithm configuration (SMAC) (Hutter et al., 2011), which expands on SPO by for example adding support for categorical rather than just numeric parameters.

High-level generate-evaluate methods are a recent approach to parameter tuning. They work similarly to simple generate-evaluate methods in that they first generate a set of parameters and then evaluate them. However, unlike the simple methods that create a set of parameters that covers the entire parameter space, the high-level methods use more advanced techniques to generate a smaller set of initial parameters that are likely to perform well. Because fewer parameters need to be evaluated, these methods are less computationally expensive than the simple methods. One example of this method is the post-selection algorithm (Yuan et al., 2013).

2.3.1.2 Parameter Control

Unlike parameter tuning which selects a single set of algorithm parameters before the algorithm is executed, parameter control is the task of changing the parameters of an algorithm during its execution. The benefit of parameter control is that the algorithm's parameters can be adjusted during execution, to fit not only the problem that is being solved as a whole but rather a specific area of the problem which might require specific parameters. However, this is a more specific task, as many approaches are limited in regard to the type of parameters of algorithms they can control. This is different from parameter tuning, where methodologies are much more general.

Karafotias et al. (2015) present a review of existing parameter control techniques, categorized by the type of parameter that is controlled. They show that parameter control

techniques that can control any type of parameter, which we will refer to as general parameter control techniques, represent a minority of existing parameter control techniques. The most popular techniques are those that control either the variation parameters or the population size. de Lacerda et al. (2021) present a review of the field of general parameter control. However, even within general parameter control, most of the presented research is limited to only a specific type or several types of algorithms. For example, only to genetic algorithms or only to evolution strategy algorithms. They identify the following categories of parameter control techniques:

Parameter and component specific methods are methods for parameter control that can only be applied to a specific parameter. This includes the control of the population size of an optimization algorithm, or its selection and variation operations. Another example is the control of the fitness function, which is particularly useful for constrained optimization. One way of dealing with constraints is to assign a penalty for solutions that do not fit the constraints, and parameter control can be used to control this penalty, for example by increasing it as time progresses.

Control ensembles are methods that involve combining multiple heterogeneous parameter control techniques to control multiple parameters at the same time. The two most common examples of this are controlling both the population size and the variation operators, or controlling both the variation and the selection parameters.

Parameter independent methods are methods that can be used for any algorithm parameter. These methods generally use various machine learning approaches, such as time-series forecasting, reinforcement learning, or machine learning models trained on specific optimization problems. Another type of a parameter-independent method is self-adaptation, which encodes the algorithm parameter into the solution representation and thus allows it to be evolved along with the solution itself.

2.3.2 Algorithm Selection

Algorithm selection, the task of selecting an algorithm that is best suited to a specific problem, was first introduced in by Rice (1976). Since then, the algorithm selection problem has received a large amount of attention and is still relevant today. This is in large part due to the advancement of machine learning technologies that have shown success in this task. In this subsection, we will describe current advances in algorithm selection, as well as a number of closely related techniques.

Before describing the techniques, we will first introduce the concepts of the Virtual Best Solver (VBS) and the Single Best Solver (SBS), which are used to evaluate the performance of an algorithm selection model on a set of optimization problems. A Virtual Best Solver, sometimes also called the oracle selector, represents a hypothetical algorithm selection model that can select the optimal algorithm for any optimization problem. A Single Best Solver represents a single optimization algorithm that achieves the best overall performance on a set of optimization problems on which the performance of an algorithm system is being measured.

Kerschke et al. (2019) present a comprehensive overview of the current state of the art in algorithm selection across various domains. They describe a number of techniques that can be used to improve performance over using a single optimization algorithm. These techniques are the following:

Per-set algorithm selection aims to select an algorithm that provides the best overall performance across an entire set of optimization problems. This can be performed

by an exhaustive evaluation of all candidate algorithms, or by racing techniques similar to the ones used for parameter tuning. The optimal performance of a per-set algorithm selection model will match that of a Single Best Solver.

Per-instance algorithm selection instead aims to select the best-performing algorithm for each individual problem in a problem set. A reasonable per-instance algorithm selection model will achieve a performance that is better than a Single Best Solver, and worse or equal to the Virtual Best Solver. It is important to note that the difference between the SBS and the VBS (also called the SBS-VBS gap) is highly specific to the problem set and the algorithm portfolio used. If a single algorithm works well on a large number of problems in a given problem set, then the SBS-VBS gap will be small. On the other hand, if every problem is solved well by a different algorithm (i.e., there is a large performance complementarity between the different algorithms), then the SBS-VBS gap will be large, and we can expect the per-instance algorithm selection model to provide a performance increase over the per-set algorithm selection model.

A closely related task to algorithm selection is the task of algorithm performance prediction. In this case, instead of selecting the best possible algorithm, the model attempts to predict the performance of each individual algorithm. The task of performance prediction can be used for algorithm selection by simply selecting the algorithm for which the model predicted the best performance. However, performance prediction models give us additional information, as they provide information about the performance of every algorithm, rather than just selecting the best one.

In the field of numerical optimization, this type of algorithm selection has achieved only limited attention. Kerschke et al. remark that "In single-objective continuous optimization, only a few studies directly and successfully address the algorithm selection problem in an automated way" (Kerschke et al., 2019, p.22).

Parallel algorithm portfolios run several different optimization algorithms, and then select the result of the best performing one. This achieves better results than using a single algorithm, but running multiple algorithms is more computationally expensive. The additional computational requirements can be mitigated by running the algorithms in parallel, or by using other techniques, such as sharing the population across all algorithms in order to reduce the number of function evaluations (Vrugt & Robinson, 2007).

Algorithm schedules similarly run several different algorithms. However, unlike parallel portfolios, algorithm schedules run the algorithms sequentially, with each algorithm using the final population of the previous one. These schedules can be static, with the order of algorithms defined before execution. Or they can be dynamic, with the algorithms and the budgets allocated to each algorithm determined during execution. One example of this approach is provided by Lindauer et al. (2016)

In recent research, exploratory landscape analysis has shown promising results when used for algorithm selection and algorithm performance prediction. For example, Kerschke and Trautmann (2019a) were able to reduce the estimated runtime of the optimization procedure by half compared to only using a single algorithm by selecting a specific algorithm to solve individual problems of the BBOB problem set.

However, some literature has raised concerns over the generalizability of landscape features when applied to algorithm selection, specifically whether the knowledge obtained from landscape features computed on one set of problems is generalizable to a different set

of problems. However, research into this topic is still fairly limited. One example of such research is presented by Lacroix and McCall (2019), which shows poor results when using data learned from a set of artificially generated problems called interpolated continuous optimization problems (ICOP) to predict the best algorithm on the set of BBOB problems. However, in the presented experiment, the approach used performed poorly even when trained and evaluated solely on the ICOP problems, which makes performance assessment a harder task.

Chapter 3

Performance Space Analysis

The performance space of optimization describes how well optimization algorithms perform on optimization problems. It is concerned with measuring algorithm performance through various metrics, with the goal of designing the best performing algorithms possible. Compared to the problem space, it has received a relatively greater amount of scientific attention, with additional algorithms being developed every year. However, in practice, a large amount of these algorithms are only incremental iterations of existing algorithms, with relatively few entirely new ideas being proposed (Aranha et al., 2022; Molina et al., 2018). Further, the evaluation of algorithm performance is mainly done through optimization benchmarks: relatively small sets of hand-picked benchmark problems. This makes it hard to determine how the performance measured by these benchmarks is reflected when solving practical problems, and how well it reflects overall algorithm performance rather than just fine-tuning the performance on the specific set of benchmark problems.

In this chapter, we examine the performance space of numerical black-box optimization. We present the analysis of two widely used numerical optimization benchmarks: the benchmarks used by the CEC Special Sessions and Competitions on Real-Parameter Single Objective optimization (CEC competitions), and the benchmark used by the GECCO Black-Box Optimization Benchmark workshops (BBOB workshops). Specifically, we are interested in determining whether there are differences in algorithm performance between the two benchmarks, in determining the improvements in performance over the years the benchmark events have been run, and in determining whether the benchmark events have contributed to an improvement in overall algorithm performance.

The first goal of this analysis is to compare the performance of different benchmark algorithms using deep statistical comparison. We would also like to discover if there is a single best-performing algorithm on these particular sets of benchmark problems. The second, more important, goal is to investigate whether there has been an improvement in overall algorithm performance throughout the years that the two benchmarks have been used to evaluate algorithm performance, or if the algorithm rankings of the benchmarks are simply a result of the algorithms performing well on a specific problem set. The CEC competitions in particular vary the set of problems between years and only compare the results of the algorithms from each individual year. Because of this, it is hard to determine whether algorithms from more recent years outperform algorithms from previous years.

The final goal is to briefly analyze the performance of different optimization algorithms across the two benchmarks. That is, to analyze the performance of the algorithms that were entered into the CEC competitions on the problem set used by the BBOB benchmarks and vice versa. With this, we aim to determine if the algorithms that were found to perform the best on one benchmark problem set will also perform well on the other. Because of the difference in the benchmark design and evaluation methodology, we expect that the

algorithms will perform worse when tested on a different benchmark. This is because, for example, the algorithms from the CEC competitions (CEC algorithms) are not designed to prioritize the speed of convergence which is evaluated by the BBOB workshops, while the algorithms designed for the BBOB benchmarks (BBOB algorithms) might not perform well on higher dimensions such as $D = 100$, which are not evaluated in the BBOB workshops. We will refer to this analysis between two benchmark sets as a cross-benchmark analysis.

3.1 Methodology

In this section, we present an overview of the methodology used for the comparisons. We first describe the problems and algorithms that were used for the comparisons. We then present the two different analysis methods that were used to compare the performance of algorithms: one using empirical cumulative distribution functions, and one that uses deep statistics. These comparisons are performed in multiple ways, for example across different dimensions, or different years of the benchmarks, in order to determine how the algorithms perform on different dimensions, and on different problem sets.

The basic overview of our experiment is the following:

1. Select the winning algorithm of each benchmark, or winners in case of a tie.
2. Obtain the performance data of the selected algorithms, either by running their source code (CEC competitions) or directly using publicly available data (BBOB workshops).
3. Evaluate the results using the Deep Statistical Comparison approach (Eftimov, Petelin, et al., 2020). In the case of BBOB, additionally evaluate using the ECDF approach.

3.1.1 Algorithms and Problems

The first set of problems we will examine in this chapter are the problems used by CEC competitions (CEC Problems). These competitions vary their problem sets between the years and only provide result data for the single year that each of the algorithms entered the competition. Since every year can use a different set of benchmark problems, the official results data cannot be used to compare algorithms from different years. In order to compare the algorithms, they need to be rerun on the benchmark problems from every competition year. We performed these additional computations using the source code of the algorithms, which is made available online by the competition organizers (J.-J. Liang et al., 2021). Due to the large number of algorithms that have entered the competitions throughout the years, we limited our analysis to only the top-performing algorithm from each year.

The second set of problems we will examine in this chapter are the ones used by the BBOB workshops (BBOB problems). Unlike the CEC problems, the BBOB workshops have not changed their original set of 24 noise-free base benchmark problems. Instead, every year uses a different set of problem instances. However, these should not greatly alter the performance of optimization algorithms, with the authors stating that "The underlying assumption when analyzing the data is that different instances of the same test function have similar difficulties" (Hansen et al., 2021). Because of this, we will focus on examining just the 24 problems and not the individual instances, by disregarding the differences between individual instances of the same problem. The performance data of all algorithms is provided on the workshops' website ("BBOB algorithm data," 2018). No additional computations are required to collect the algorithm data, as the provided data

uses the same format for every year of the workshops, which makes it directly comparable. We chose to focus our comparison on the algorithms that have entered the workshops in the period between the years 2009, when the workshops were first held, and 2018, when the analysis described in this chapter was performed. During this period, a total of 196 algorithms submitted their data. Due to a large number of algorithms, we chose to limit ourselves to only a number of best-performing algorithms from each year, like we did with the CEC competitions.

Finally, we will compare the performance of the CEC algorithms evaluated on the 24 BBOB problems, and the performance of the BBOB algorithms on the problems from the 2017 CEC problem set. We will use the algorithms that performed well on their respective benchmarks. However, unlike the CEC competitions, the BBOB workshops do not require submitted algorithms to publish their source code. As a result, only a small fraction of algorithms listed on the organizer's website ("BBOB algorithm data," 2018) provide their source code. This limits which algorithms we can use for our analysis.

3.1.2 Deep Statistical Analysis

The benchmark analysis was performed using Deep Statistical Comparison (DSC) approach (Eftimov, Korošec, & Seljak, 2017). The data involved in the multiple-problem analysis for each comparison is obtained by using the DSC ranking scheme, which is based on the whole distribution of the results, instead of using only one statistic to describe the distribution, such as mean or median, which is important for reproducible analysis. DSC removes the sensitivity of the simple statistics to the data found in common approaches (Derrac et al., 2011; García et al., 2009) and enables us to calculate more robust statistics without fearing the influence of outliers or some errors inside ϵ-neighborhood. Further, the data is analyzed using an appropriate omnibus statistical test, i.e., a statistical test which tests more than two independent variables to determine if there is a difference between the variables involved, but cannot test the differences between individual variables.

In our experiments, we performed independent pairwise comparisons between each pair of algorithms. We used a left one-tailed omnibus statistical test. A one-tailed test allows us to determine if one value is greater or less than another value, but not both. If the sample being tested falls into the one-sided critical area, the alternative hypothesis will be accepted instead of the null hypothesis. In our case, the alternative hypothesis statement contains a lower than ($<$), which means that it will be accepted if the first algorithm has better performance than the second one. After checking the required conditions for the safe use of the parametric tests (i.e., normality of the data and homoscedasticity of the variances), the Wilcoxon signed-rank test was selected as the appropriate omnibus statistical test for all involved comparisons.

3.1.3 Benchmarking Scenarios

For the CEC competitions, we performed three different types of comparisons. First, we compared the algorithm performances between the different years of the competition, and thus across the different problem sets. With this, we hoped to see if there has been measurable improvement in performance throughout the years of the competition. In other words: can more recently proposed algorithms solve problems better than older algorithms? Second, we compared them by the dimension of the problems being solved. Here, we wanted to examine the trends in the performance of algorithms. For example, if more recent algorithms focus more on higher dimensions, or if the performance of algorithms on low dimensional problems has improved over the years. Finally, we compared the results on a combined benchmark set that contained all problems from all years and dimensions

in order to compare overall algorithm performance and determine whether there is a single best performing algorithm overall. For the BBOB workshop comparisons, we omitted the comparisons between the different years, as the BBOB workshops only use a single base problem set regardless of the year and only introduce additional instances, which do not significantly alter the properties of the base problems, making such a comparison unnecessary.

In the final part of this chapter, we perform a cross-benchmark analysis, where the algorithms that were submitted to one of the CEC competitions are evaluated on the BBOB benchmark and vice versa.

3.2 Experimental Setup

In this section, we give a more in-depth description of all of the technical and implementation details used in our experiments. We first present the specific problems and algorithms used in our comparison, as well as the procedure for choosing the algorithms that we will use in our experiments. We additionally provide a more detailed look at the problems chosen for our analysis.

3.2.1 Congress on Evolutionary Computation Competitions

From the CEC competitions, we analyzed the algorithm from the competitions that were held in the years 2013, 2014, 2016, 2017, and 2018. We decided to skip algorithms from the competition held in the year 2015 because that year was focused on learning-based optimization, which was not the case in other years.

For every year, we chose the algorithm that won the corresponding competition according to the official results. We chose six algorithms: one from every year, except for 2016. In 2016, two algorithms produced similar results and were declared joint winners by the competition organizers. In this case, we picked both of the winning algorithms for our evaluation. The algorithms chosen were: NBIPOPaCMA (Loshchilov, 2013) (from 2013), LSHADE (Tanabe & Fukunaga, 2014) (from 2014), L-SHADE_EpSin (N. H. Awad et al., 2016) (from 2016), UMOEAII (Elsayed et al., 2014) (from 2016), EBOwithCMAR (Kumar et al., 2017) (from 2017), and HSES (G. Zhang & Shi, 2018) (from 2018).

Table 3.1: Algorithms and parameters used for the CEC competition comparisons, with the year of competition and benchmark problems they were first evaluated on.

Algorithm	**Competition**	**Problems**
NBIPOP-aCMA (Loshchilov, 2013)	2013	2013
LSHADE (Tanabe & Fukunaga, 2014)	2014	2014
L-SHADE_EpSin (N. H. Awad et al., 2016)	2016	2014
UMOEAII (Elsayed et al., 2014)	2016	2014
EBOwithCMAR (Kumar et al., 2017)	2017	2017
HSES (G. Zhang & Shi, 2018)	2018	2017

The problem sets used for our comparisons do not entirely come from the same years as the chosen algorithms. This is because the competitions held in 2016 and 2018 did not use new benchmark problems. The competition held in 2016 used the problems from the year 2014 and the competition held in 2018 used the problems from the year 2017. In addition, while we did not use algorithms from the 2015 competition, we still included the

competition's benchmark problems. We were able to do this because the source code of the 2015 problem definitions was structured in the same way as the other competitions, making it compatible with the algorithms we chose to compare. In summary, we used the benchmark problems from the years 2013 (J. J. Liang, Qu, Suganthan, & Hernández-Díaz, 2013), 2014 (J. J. Liang, Qu, & Suganthan, 2013), 2015 (J. J. Liang et al., 2014), and 2017 (N. Awad et al., 2016) of the competitions. We note that although the problem sets are different overall depending on the year, they still include a number of overlapping individual problems, as more recent benchmarks reuse problems from previous years. Table 3.2 shows the full amount of shared problems between the different problem sets. The 2015 problem set in particular only includes 15 problems in total, with 10 of these problems being shared with the 2014 problem set, meaning that there are only 5 problems that do not appear in any other year. We still chose to include the full problem set from each year, including the shared problems. This was done to ensure more reliable statistical results. Especially for the 2015 problem set, where we would otherwise only be using the 5 unique problems to compare 6 different algorithms.

Table 3.2: Number of shared benchmark problems in different years of competition.

	2014	2015	2017
2013	10	5	7
2014	-	10	5
2015	-	-	3

All of the selected problem sets structure their problem definitions in a similar way. Because of this, most algorithms we selected were compatible with all problem sets without requiring any significant changes to their source code. The algorithms could be made compatible simply by changing the source file that contains the problem definitions from each year. The only exception was the algorithm from 2013 (NBIPOPaCMA) which also required small changes to the way it displays results to make the results comparable with other algorithms because the formatting of the results was changed with the 2014 competition. We never modified any part of the algorithm's logic and all parameters were left as originally provided. To verify that the algorithm source code worked correctly, we ran each algorithm on the benchmark set from the year it was submitted to the competition and then compared the distribution of our results with the distribution of the official competition results using a two-sample Kolmogorov-Smirnov test.

3.2.2 Black-Box Optimization Benchmarking Workshops

Because the BBOB workshops are not a competition and are focused more on understanding the performance and convergence of algorithms, they do not rank the algorithms submitted to the workshops and do not declare a single algorithm as a winner. However, it was still desirable for our research to focus on only a small subset of all algorithms submitted to the workshops. To determine which algorithms to focus on, we first used the officially published workshop results ("BBOB algorithm data," 2018) to create an ECDF analysis for each year, using the ECDF methodology. We then selected a set of algorithms that showed good performance for every year individually. For some years, this number was larger, as several algorithms showed similar performance. As a result, we chose to include all such algorithms instead of arbitrarily limiting ourselves only to the best one. Some algorithms also included data for multiple variants of the same algorithm. In such

cases, we included every variant in our analysis. Table 3.3 shows all selected algorithms.

From the table, it is apparent that the majority of selected algorithms are from earlier years of the workshops (2009 to 2012). The reason for this is the relative lack of competitive algorithms submitted in the more recent years of the workshops in terms of overall ECDF performance. This could be explained by recent algorithms focusing more heavily on a smaller subset of problems, or by algorithm developers being more focused on other BBOB problem sets, such as the noisy or the multi-objective problem sets. In some cases, the more recent years included only a single algorithm that showed overall ECDF results competitive with the earlier years.

We also did not choose any algorithms from 2016. This year introduced a new multi-objective problem set, and most algorithms submitted to the workshops focused on it instead of the single objective problem set. Only a single algorithm (with variants) was entered into the single objective workshop, and it did not include data for 40-dimensional problems. As we used the official data provided by the workshop organizers, this means that each algorithm was evaluated using the default set of parameters specified in their respective articles.

Table 3.3: Algorithms chosen for the ECDF comparison of the BBOB workshops, with the year in which the algorithm entered the competition.

Algorithm	**Year**
IPOP-SEP-CMA-ES (Ros, 2009)	2009
BIPOP-CMA-ES (Hansen, 2009)	2009
iAMALGAM (Bosman et al., 2009)	2009
NELDERDOERR (Doerr et al., 2009)	2009
AMALGAM (Bosman et al., 2009)	2009
VNS (García-Martínez & Lozano, 2009)	2009
DE-F-AUC (Fialho et al., 2010)	2010
DEuniform (Fialho et al., 2010)	2010
IPOP-ACTCMA-ES (Hansen & Ros, 2010)	2010
IPOP-CMA-ES (Hansen & Ros, 2010)	2010
MOS (LaTorre et al., 2010)	2010
JADE (Pošík & Klemš, 2012a)	2012
JADEb (Pošík & Klemš, 2012b)	2012
JADEctpb (Pošík & Klemš, 2012b)	2012
NBIPOPaCMA (Loshchilov et al., 2012b)	2012
NIPOPaCMA (Loshchilov et al., 2012b)	2012
BIPOP-aCMA-STEP (Loshchilov et al., 2013)	2013
CMA-CSA (Atamna, 2015)	2015
CMA-MSR (Atamna, 2015)	2015
CMA-TPA (Atamna, 2015)	2015
KL-BIPOP-CMA-ES (Yamaguchi & Akimoto, 2017)	2017
KL-IPOP-CMA-ES (Yamaguchi & Akimoto, 2017)	2017
KL-Restart-CMA-ES (Yamaguchi & Akimoto, 2017)	2017
PSA-CMA-ES (Nishida & Akimoto, 2018)	2018
PSA-CMA-ESwRS (Nishida & Akimoto, 2018)	2018

As the BBOB workshops use a single set of 24 problems, all of these problems were used in our analysis. For each problem, 15 instances were used. However, the specific instances

used differed with the years of the benchmarking workshops. As the ECDF analysis showed that algorithms performed similarly in lower dimensions (see Section 3.3.2.1), we chose to focus our deep statistical analysis on the higher dimensions of $D = 5$, $D = 10$, $D = 20$, and $D = 40$, while ignoring the dimensions of $D = 2$ and $D = 3$.

We note that due to the experimental setup of the BBOB workshops, not all of the algorithms chosen use the same evaluation budget in a single algorithm run. This can be explained by the fact that the BBOB experimental setup allows for algorithm restarts, and thus stopping an algorithm's execution prematurely can lead to improved overall performance. However, our deep statistical analysis requires a fixed budget under which the algorithms are compared. We account for this difference in experimental design by performing multiple comparisons using two different evaluation budgets, 10^6 and 10^5, and by noting which algorithms did not use the full evaluation budget. Additionally, the BBOB workshops use a number of different problem error targets, from 100 to 10^{-8}. In our analysis, we will use a single target value of 10^{-8}.

3.3 Results

In this section, we present the results of all the comparisons that were performed. We group the results first by the benchmark event (either the CEC competitions or the BBOB benchmarks), and then by the types of benchmark scenarios that were analyzed.

3.3.1 Congress on Evolutionary Computation Competitions

The results of the CEC competition evaluations are split into three benchmark scenarios representing the groupings described previously: by year, by dimension, and across the combined benchmark set. In all of the tables representing results, the algorithm names have been shortened due to table width constraints. The list of abbreviations is available in Table 3.4. Each abbreviation contains two digits that describe the year when the algorithm entered the competition and the first letter of the algorithm name.

While evaluating the results, we paid close attention to the year when each algorithm entered the competition. We expected algorithms to perform better in these years, as they were possibly tuned on these problems. An overview of the competition years and their corresponding benchmark sets are shown in Table 3.1. Note that the year the algorithm was published does not always correspond to the benchmark set that the algorithm was first evaluated on, as in some years the CEC competition reused older benchmark problem sets.

Table 3.4: Algorithms used in results tables.

Algorithm	Abbreviation
NBIPOPaCMA	13-N
LSHADE	14-L
LSHADE_EpSin	16-L
UMOEA	16-U
EBO	17-E
HSES	18-S

For every benchmark set, we present the results in a single table representing pairwise comparisons. Before a statistical test was applied, the data was ranked using the DSC

ranking scheme (Eftimov, Korošec, & Seljak, 2017). Afterward, the ranked data was analyzed using the Wilcoxon signed-rank test. Every table cell represents the p-value obtained when comparing two algorithms. A p-value lower than 0.05 shows that the algorithm in the row performed *better* than the algorithm in the column. For visibility, cells with such p-values have a gray background. All p-values have been rounded to two decimal places.

3.3.1.1 Results When Grouping by Year

First, we present the results of our analysis, grouped by year. This grouping is used to show if there was any significant performance increase during the years of the competition. It is also used to examine how much specific algorithms might be fitted to a specific year of the competition. In the tables that describe the results in this Section, an asterisk after an algorithm's name shows which algorithms were originally developed for this year's benchmark problems.

The results for the benchmark set from the year 2013 presented in Table 3.5 show that NBIPOPaCMA performed worse than every other algorithm, despite the fact that it entered the competition this year. Surprisingly, as we will see later, NBIPOPaCMA performed much better in other benchmark sets. HSES, from 2018, performed better than every other algorithm tested. We interpret this to indicate that there has been improvement in solving the 2013 benchmark set throughout the years of the competition.

Table 3.5: CEC – Results for the benchmark set from the year 2013.

	13-N	14-L	16-L	16-U	17-E	18-H
13-NBIPOPaCMA*	-	1.00	1.00	1.00	1.00	1.00
14-LSHADE	0.00	-	0.99	0.78	0.02	1.00
16-LSHADE_EpSin	0.00	0.00	-	0.04	0.00	0.97
16-UMOEA	0.00	0.26	0.96	-	0.00	1.00
17-EBO	0.00	0.97	1.00	1.00	-	1.00
18-HSES	0.00	0.00	0.03	0.00	0.00	-

The results for the benchmark set from the year 2014 presented in Table 3.6 show that the two algorithms from 2016 (LSHADE_EpSin and UMOEA) performed well, with UMOEA doing better than LSHADE_EpSin. It is important to note that both algorithms from 2016 could have been tuned for this benchmark set, as the 2016 benchmark had no new problems.

The results from the year 2015 are presented in Table 3.7. Here, UMOEA and EBO show good results, with both of them outperforming the two LSHADE algorithms.

In 2017, the last problem set was introduced. Both HSES and EBO were tuned for this benchmark set, as the 2018 competition reused the 2017 set. The results from this year are presented in Table 3.8. Surprisingly, NBIPOPaCMA performed better than both LSHADE variants, showing results similar to EBO and HSES.

While the article for the EBO algorithm (Kumar et al., 2017) showed massive improvements over LSHADE_EpSin and UMOEA in this benchmark set, we could not reproduce these results. In our own results, despite using the same source code and algorithm parameters, EBO performed better than LSHADE_EpSin (as the article indicated), but could not outperform UMOEA, with a p-value of 0.13. This could be due to several reasons. One is the inherently stochastic nature of the algorithm. This could also be a result of the

Table 3.6: CEC – Results for the benchmark set from the year 2014.

	13-N	14-L	16-L	16-U	17-E	18-H
13-NBIPOPaCMA	-	0.34	1.00	1.00	0.46	0.06
14-LSHADE*	0.65	-	0.94	1.00	1.00	0.28
16-LSHADE_EpSin*	0.00	0.06	-	0.91	0.07	0.01
16-UMOEA*	0.00	0.00	0.08	-	0.00	0.00
17-EBO	0.53	0.00	0.93	1.00	-	0.03
18-HSES	0.94	0.72	1.00	1.00	0.97	-

Table 3.7: CEC – Results for the benchmark set from the year 2015.

	13-N	14-L	16-L	16-U	17-E	18-H
13-NBIPOPaCMA	-	0.29	0.33	0.76	0.66	0.81
14-LSHADE	0.71	-	0.62	0.98	1.00	0.91
16-LSHADE_EpSin	0.66	0.38	-	0.99	0.99	0.71
16-UMOEA	0.23	0.01	0.01	-	0.82	0.09
17-EBO	0.33	0.00	0.00	0.18	-	0.13
18-HSES	0.18	0.09	0.29	0.91	0.88	-

different statistical methods used to compare algorithms' performances. Unfortunately, the article for EBO (Kumar et al., 2017) does not provide a detailed overview of the statistical methods used in the comparison. However, the comparison method used in the article only compares the mean values of the distribution of the results, while we compare using the entire distribution of the results. We believe that the algorithm code we used to generate the results performed correctly, as a two-sample Kolmogorov–Smirnov test between our results and the official 2017 competition results showed our data to be consistent with the official results.

Table 3.8: CEC – Results for the benchmark set from the year 2017.

	13-N	14-L	16-L	16-U	17-E	18-H
13-NBIPOPaCMA	-	0.02	0.02	0.79	0.66	0.90
14-LSHADE	0.98	-	0.24	1.00	1.00	0.99
16-LSHADE_EpSin	0.98	0.76	-	0.54	1.00	1.00
16-UMOEA	0.20	0.02	0.45	-	0.86	0.82
17-EBO*	0.34	0.00	0.00	0.13	-	0.69
18-HSES*	0.09	0.01	0.00	0.18	0.30	-

With the exception of NBIPOPaCMA and the year 2013, there is a trend that every algorithm performs well on the benchmark set from the year in which it entered the competition. In addition, it is clear that the two algorithms from 2016 perform better than the one from 2014 (they were all tuned for the same benchmark set from the year 2014). This relation does not hold for the algorithms from the 2017 and 2018 competitions (EBO

and HSES), both of which were similarly tuned for the 2017 benchmark set. In the years 2014 and 2015, EBO in a larger number of cases outperforms more algorithms than HSES, while the reverse is true for 2013 and 2017.

3.3.1.2 Results When Grouping by Dimension

The second grouping of benchmark problems was by dimension, where we combined problems of all of the years of the competitions into a single set, separated by problem dimension. Throughout the years of the competitions, some of the benchmark problems were reused in different years. To ensure unbiased results, we made sure to remove duplicate problems when merging benchmark sets of different years, so that each problem was used exactly once.

With this grouping, we wanted to see trends throughout the years with regard to problem dimension. For example, newer algorithms might focus more on higher-dimensional problems where $D \geq 50$ if old algorithms are already good at solving lower-dimensional ones where $D < 50$.

For the dimension of $D = 10$, the results are shown in Table 3.9. UMOEA performed better than every other algorithm. The official results of the 2016 competition showed that it outperformed the other winner from that year (LSHADE_EpSin) in lower dimensions (Suganthan et al., 2016). Our results are consistent with those results, as they show that UMOEA outperforms all other winners of the competition when it comes to 10-dimensional problems.

Table 3.9: CEC – Results for 10-dimensional problems.

	13-N	14-L	16-L	16-U	17-E	18-H
13-NBIPOPaCMA	-	1.00	1.00	1.00	0.82	0.80
14-LSHADE	0.00	-	0.01	1.00	0.02	0.00
16-LSHADE_EpSin	0.00	0.99	-	1.00	0.15	0.00
16-UMOEA	0.00	0.00	0.00	-	0.00	0.00
17-EBO	0.18	0.98	0.85	1.00	-	0.20
18-HSES	0.2	1.00	1.00	1.00	0.79	-

At the dimension of $D = 30$, shown in Table 3.10, we can see that UMOEA and EBO perform similarly and outperform most algorithms, while NBIPOPaCMA performs worse than every other algorithm. For the dimension of $D = 50$, the results are shown in Table 3.11. Here, we can see that LSHADE_EpSin starts performing slightly better than in lower dimensions. LSHADE_EpSin, UMOEA, and EBO all performed well. NBIPOPaCMA once again performed poorly. The results for the dimension of $D = 100$ are shown in Table 3.12. LSHADE_EpSin and HSES are the clear winners, outperforming every other algorithm except each other. This is consistent with previous evaluations of both algorithms (N. H. Awad et al., 2016; J.-J. Liang et al., 2021; G. Zhang & Shi, 2018), where it was shown that they perform well for higher dimensional problems.

In general, this shows that UMOEA is the best of the evaluated algorithms for lower dimensional problems, while LSHADE_EpSin and HSES are the best at higher dimensional problems. The algorithms from the years 2013 and 2014 do not excel in any particular dimension.

Table 3.10: CEC – Results for 30-dimensional problems.

	13-N	14-L	16-L	16-U	17-E	18-H
13-NBIPOPaCMA	-	0.99	1.00	1.00	0.99	0.96
14-LSHADE	0.01	-	0.83	1.00	1.00	0.50
16-LSHADE_EpSin	0.00	0.17	-	0.78	0.72	0.21
16-UMOEA	0.00	0.00	0.23	-	0.34	0.02
17-EBO	0.01	0.00	0.28	0.67	-	0.04
18-HSES	0.04	0.50	0.79	0.98	0.96	-

Table 3.11: CEC – Results for 50-dimensional problems.

	13-N	14-L	16-L	16-U	17-E	18-H
13-NBIPOPaCMA	-	0.87	1.00	1.00	0.98	1.00
14-LSHADE	0.13	-	1.00	1.00	1.00	0.95
16-LSHADE_EpSin	0.00	0.00	-	0.09	0.12	0.73
16-UMOEA	0.00	0.00	0.91	-	0.77	0.93
17-EBO	0.02	0.00	0.88	0.23	-	0.72
18-HSES	0.00	0.05	0.27	0.06	0.15	-

Table 3.12: CEC – Results for 100-dimensional problems.

	13-N	14-L	16-L	16-U	17-E	18-H
13-NBIPOPaCMA	-	0.93	1.00	1.00	0.98	1.00
14-LSHADE	0.07	-	1.00	0.98	1.00	1
16-LSHADE_EpSin	0.00	0.00	-	0.00	0.01	0.90
16-UMOEA	0.00	0.01	1.00	-	0.84	1.00
17-EBO	0.01	0.00	0.98	0.16	-	1.00
18-HSES	0.00	0.00	0.10	0.00	0.00	-

3.3.1.3 Results for All Problems

For the final grouping, we merged benchmark problems from all benchmark sets into one complete set, using problems from all years on all dimensions. Like in the grouping by dimension, some benchmark problems appeared multiple times, as they were reused in different years of the competition. All such duplicates were removed so that each problem would be evaluated only once. This grouping is used to show whether any of the algorithms can claim to be the best in general with regard to all benchmark problems and if there have been any general improvements throughout the years of the competition.

The results for all problems on all dimensions are shown in Table 3.13.

Table 3.13: CEC – Results for all benchmark problems.

	13-N	14-L	16-L	16-U	17-E	18-H
13-NBIPOPaCMA	-	1.00	1.00	1.00	1.00	1.00
14-LSHADE	0.00	-	1.00	1.00	1.00	0.93
16-LSHADE_EpSin	0.00	0.00	-	0.47	0.02	0.22
16-UMOEA	0.00	0.00	0.54	-	0.04	0.18
17-EBO	0.00	0.00	0.99	0.96	-	0.84
18-HSES	0.00	0.07	0.77	0.82	0.15	-

In the combined benchmark set, we can see that both algorithms from 2016 performed well. Both outperformed every algorithm except for HSES and each other. This seems consistent with previous benchmark sets, where both of these algorithms showed good results. The algorithms from 2013 (NBIPOPaCMA) and 2014 (LSHADE) performed poorly.

The algorithms from the years 2017 (EBO) and 2018 (HSES) both performed worse overall than the algorithms from 2016. This shows that overall, there has been no significant improvement in algorithm performance on the CEC benchmark problems.

3.3.2 Black-Box Optimization Benchmarking Workshops

We performed a similar analysis using the BBOB problems. However, unlike the CEC competitions, the BBOB workshops use a single set of base benchmark problems and only vary the problem set by introducing additional problem instances, which do not greatly alter the properties of the underlying base problem. Because of this, it is not possible to create a comparison of the algorithm performance between the different years as we have done with the CEC problems. Instead, our analysis focuses on determining whether there is an increase in overall algorithm performance and whether there is a statistically significant difference between the performance of the most successful algorithms of the benchmarking workshops.

In this section, we provide the results of our analysis. We provide the results of both the ECDF and deep statistical approaches, and highlight the similarities and differences between the obtained results. We also perform a more in-depth analysis using deep statistical comparison in order to determine why the results provided by the two approaches differ.

3.3.2.1 ECDF Results

Here, we present the results of the analysis using the ECDF approach over all benchmark problems. The ECDF results are available in Figures 3.1-3.6 for dimensions of $D = 2$,

$D = 3$, $D = 5$, $D = 10$, $D = 20$, and $D = 40$. In these figures, the line marked *best 2009* shows the performance of an artificial solver created by taking the results of the best performing algorithm for each base problem, dimension, and target individually, and combining their results as if they were a single algorithm. This shows the theoretical best performance that could be achieved if we selected for every problem, dimension, and target the most optimal algorithm from the algorithms submitted in 2009. Because of this, this artificial solver is able to achieve a performance that is better than any individual algorithm submitted in 2009.

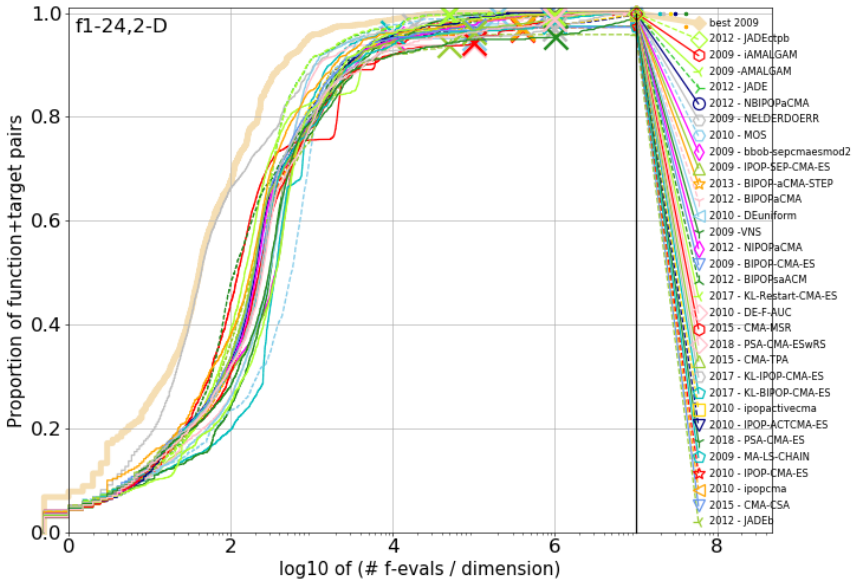

Figure 3.1: ECDF results for two-dimensional problems.

From the ECDF results, we see that the algorithm BIPOP-aCMA-STEP (Loshchilov et al., 2013) appears to be the clear winner from five-dimensional problems onward. Up to and including the dimension of $D = 20$, it trails behind BIPOPsaACM (Loshchilov et al., 2012c) when using a low number of problem evaluations but overtakes it at around $10^3 D$ problem evaluations. The BIPOPsaACM algorithm did not include results data for the dimension of $D = 40$ dimensions, so we cannot confirm that such a result would also be present in 40-dimensional problems.

Overall, looking solely at the data across all problems, the 2013 algorithm BIPOP-aCMA-STEP appears to be the best performing algorithm when problem evaluations exceed $10^3 D$. No algorithms from competitions taking place after 2013 that were included in our study show an improvement over this performance.

We can see that the relative difference between the algorithms increases with dimension. In low dimensions, most algorithms show similar behavior: they solve most problems successfully. As dimension increases, fewer and fewer algorithms are able to solve the problems. In general, this seems to indicate that either the lower dimensions are pretty well understood by algorithm authors, or that they are too easy to solve by currently used optimization algorithms. In both cases, it seems they do not serve much purpose when it comes to comparing algorithm performance. Because of this, we chose to limit the deep statistical analysis to dimensions of $D = 5$ and higher.

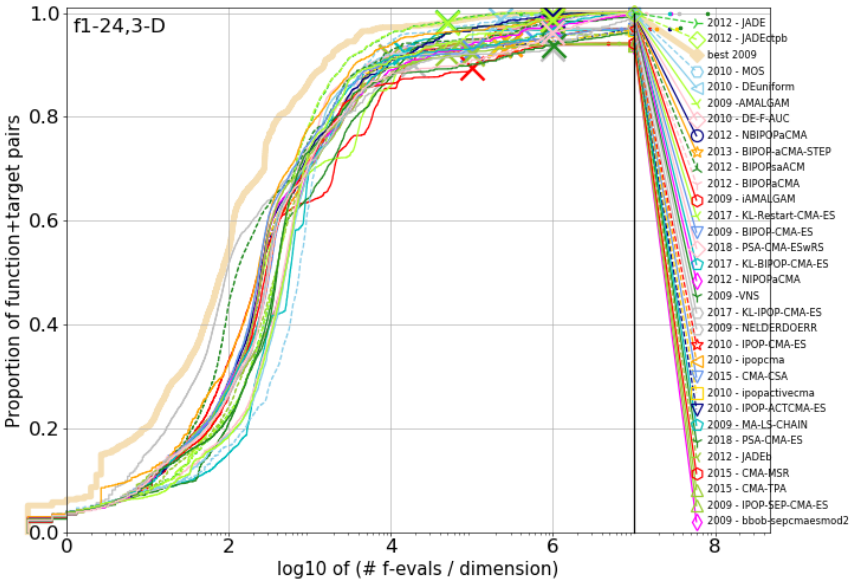

Figure 3.2: ECDF results for three-dimensional problems.

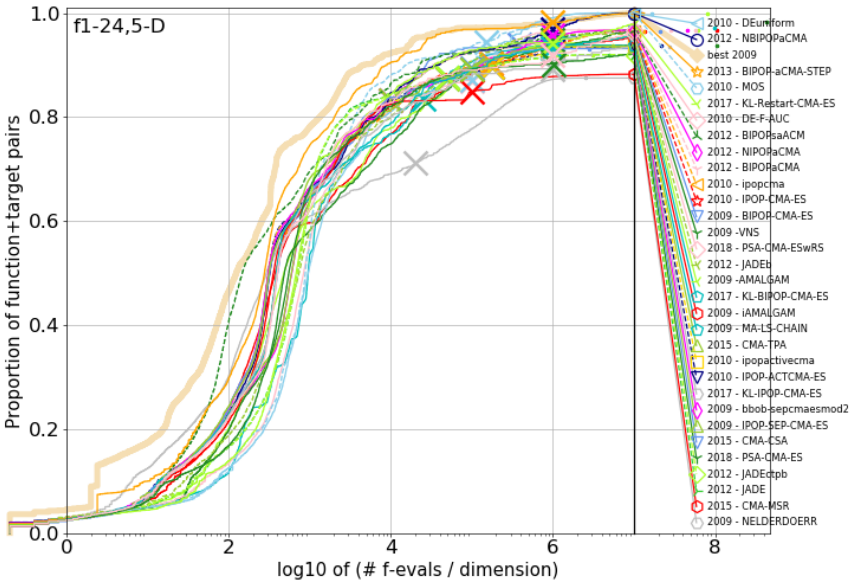

Figure 3.3: ECDF results for five-dimensional problems.

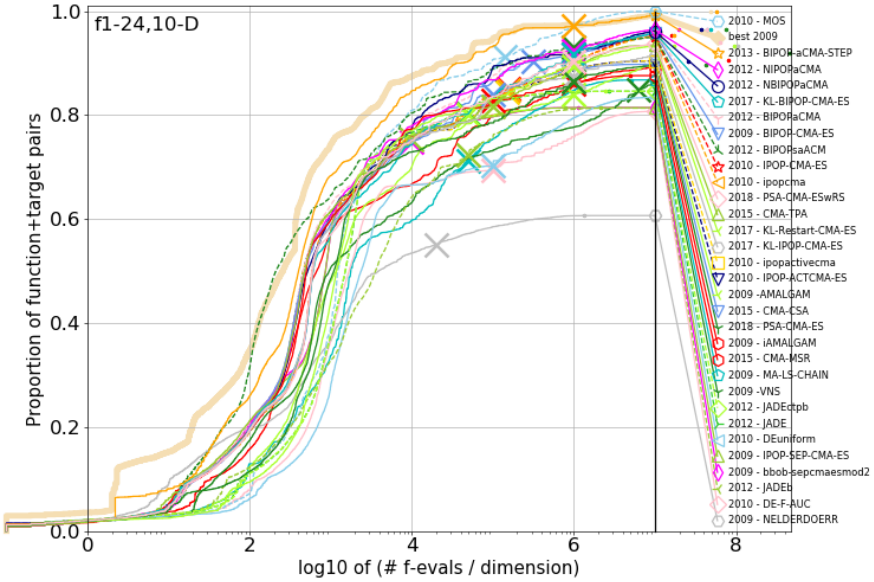

Figure 3.4: ECDF results for 10-dimensional problems.

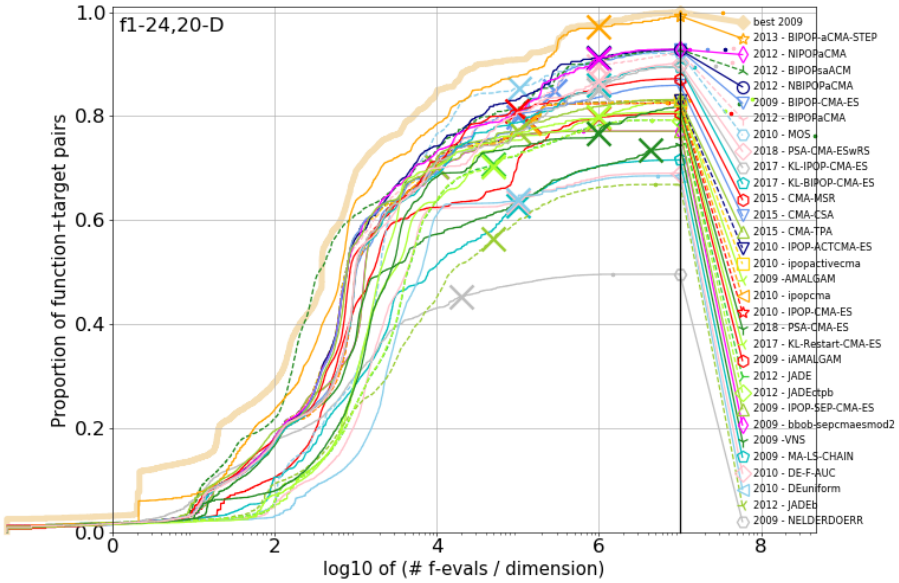

Figure 3.5: ECDF results for 20-dimensional problems.

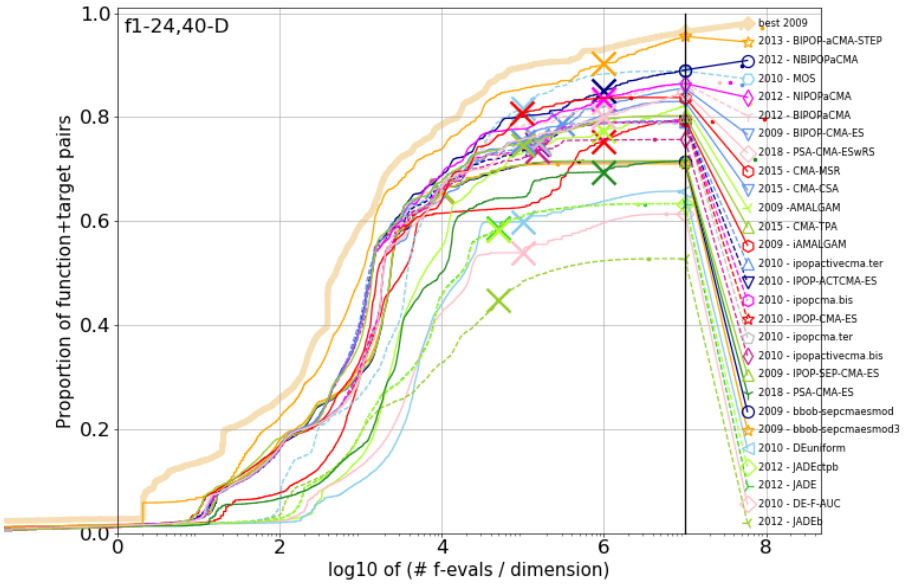

Figure 3.6: ECDF results for 40-dimensional problems.

3.3.2.2 Deep Statistical Results

The deep statistical results are presented in two ways. Table 3.14 presents the results of all of the analyzed algorithms. For these results, we do not present the direct comparison of every algorithm versus every other algorithm, as the resulting table would be too large and unreadable. Instead, for every algorithm, we present the number of algorithms for which the given algorithm showed statistically significantly better performance. These results are presented by dimension, with each column representing a single dimension. Some algorithms did not provide data for 40-dimensional problems, so they are marked with a - in that column.

The table shows results that are consistent with the ECDF analysis. For every dimension, BIPOP-aCMA-STEP (Loshchilov et al., 2013) statistically outperforms the most algorithms considering counting of independent pairwise comparisons, with NIPOPaCMA (Loshchilov et al., 2012a), NBIPOPaCMA (Loshchilov et al., 2012a), and BIPOP-CMA-ES (Hansen, 2009) showing similar performance. However, BIPOP-aCMA-STEP is not statistically better than every other algorithm (i.e., does not show statistically significant performance in each independent pairwise comparison) like the ECDF analysis would suggest.

Next, we examine the results of directly comparing a small number of hand-selected algorithms. As when presenting the results of the CEC competitions, the algorithm names in table headers have been abbreviated. The list of abbreviations used is shown in Table 3.15. The abbreviations are constructed by taking the last two digits of the year when the algorithm entered the competition, combined with the first two letters of the algorithm name. Tables 3.16, 3.17, 3.18, and 3.19 show the results of this analysis. Here, algorithms are compared against each other using pairwise statistical comparison. The values in the cells show the p-values of these comparisons. A value of < 0.05 means that the algorithm in the row of the table performed statistically better than the algorithm in the column. For easier visualization, such cells have a darker background. As in the previous

Table 3.14: Results of the deep statistical analysis using full results data with $10^6 D$ problem evaluations, sorted by the year. Each column shows the number of algorithms a given algorithm performed better than. Algorithms marked with an asterisk (*) did not use the full $10^6 D$ evaluations and are ranked using best available data.

Algorithm	**5D**	**10D**	**20D**	**40D**
IPOP-SEP-CMA-ES*	0	1	4	-
BIPOP-CMA-ES*	11	10	15	8
iAMALGAM*	11	6	8	5
NELDERDOERR	0	0	0	-
AMALGAM*	11	5	7	6
VNS	8	5	2	-
DE-F-AUC*	1	0	0	0
DEuniform*	1	1	0	6
IPOP-ACTCMA-ES*	0	4	7	-
IPOP-CMA-ES*	1	3	4	-
MOS*	8	8	8	-
JADE*	0	2	2	1
JADEb*	0	1	0	0
JADEctpb*	0	2	2	1
NBIPOPaCMA	11	14	17	12
NIPOPaCMA	11	14	17	11
BIPOP-aCMA-STEP	13	21	20	13
CMA-CSA*	1	5	7	5
CMA-MSR*	0	3	7	5
CMA-TPA*	0	2	5	5
KL-BIPOP-CMA-ES	6	10	9	-
KL-IPOP-CMA-ES	1	3	8	-
KL-Restart-CMA-ES	11	10	10	-
PSA-CMA-ES	2	4	3	2
PSA-CMA-ESwRS	6	9	13	6

tables, instead of comparing all algorithms directly, we instead only show a subset of the best performing algorithms. These algorithms are BIPOP-aCMA-STEP (Loshchilov et al., 2013), NIPOPaCMA (Loshchilov et al., 2012a), NBIPOPaCMA (Loshchilov et al., 2012a), BIPOP-CMA-ES (Hansen, 2009), and PSA-CMA-ESwRS (Nishida & Akimoto, 2018).

Table 3.15: Algorithm abbreviations used in results tables.

Algorithm	Abbreviation
BIPOP-CMA-ES	09-BI
NBIPOPaCMA	12-NB
NIPOPaCMA	12-NI
BIPOP-aCMA-STEP	13-BI
PSA-CMA-ESwRS	18-PS

Table 3.16: BBOB – Deep statistical results for five-dimensional problems.

	09-BI	12-NB	12-NI	13-BI	18-PS
09-BIPOP-CMA-ES	-	0.98	0.98	0.98	0.68
12-NBIPOPaCMA	0.50	-	0.68	0.98	0.17
12-NIPOPaCMA	0.50	0.68	-	0.68	0.39
13-BIPOP-aCMA-STEP	0.50	0.50	0.68	-	0.07
18-PSA-CMA-ESwRS	0.68	0.97	0.80	0.97	-

Table 3.17: BBOB – Deep statistical results for 10-dimensional problems.

	09-BI	12-NB	12-NI	13-BI	18-PS
09-BIPOP-CMA-ES	-	0.97	0.97	0.98	0.72
12-NBIPOPaCMA	0.17	-	0.68	0.97	0.38
12-NIPOPaCMA	0.17	0.68	-	0.97	0.80
13-BIPOP-aCMA-STEP	0.04	0.17	0.17	-	0.07
18-PSA-CMA-ESwRS	0.38	0.80	0.39	0.98	-

In Tables 3.16, 3.17, 3.18, and 3.19, we can see even more clearly that the deep statistical approach disagrees about the significance of the performance advantage of BIPOP-aCMA-STEP. While the ECDF graphs show that BIPOP-aCMA-STEP greatly outperforms every other algorithm (particularly at higher dimensions), this is not the result obtained from the deep statistical approach. For example, when using the deep statistical approach, there is no statistical significance between each pair of the following algorithms (NBIPOPaCMA, NIPOPaCMA), (NBIPOPaCMA, BIPOP-aCMA-STEP), and (NIPOPaCMA, BIPOP-aCMA-STEP), regardless of dimension. Furthermore, in the dimensions of $D = 5$ and $D = 20$, there is no statistical significance found between any pairs of the five algorithms used in the deep statistical comparison. On the other hand, the ECDF method shows a gap between BIPOP-aCMA-STEP and the other four algorithms

Table 3.18: BBOB – Deep statistical results for 20-dimensional problems.

	09-BI	12-NB	12-NI	13-BI	18-PS
09-BIPOP-CMA-ES	-	1.00	0.97	0.97	0.82
12-NBIPOPaCMA	0.17	-	0.80	0.58	0.38
12-NIPOPaCMA	0.5	0.97	-	0.88	0.58
13-BIPOP-aCMA-STEP	0.38	0.58	0.21	-	0.11
18-PSA-CMA-ESwRS	0.24	0.8	0.57	0.93	-

Table 3.19: BBOB – Deep statistical results for 40-dimensional problems.

	09-BI	12-NB	12-NI	13-BI	18-PS
09-BIPOP-CMA-ES	-	0.98	1.00	0.99	0.54
12-NBIPOPaCMA	0.04	-	0.58	0.58	0.02
12-NIPOPaCMA	0.01	0.58	-	0.58	0.06
13-BIPOP-aCMA-STEP	0.02	0.58	0.58	-	0.21
18-PSA-CMA-ESwRS	0.54	0.99	0.96	0.88	-

that to a casual observer would appear very significant. However, we stress here that the difference between the deep statistical and the ECDF approaches do not show that one of these approaches is superior, but simply show the differences that can arise when using different methods to examine the results.

Another interesting detail of our analysis appears in the 40-dimensional analysis, shown in Table 3.19. We can see that NBIPOPaCMA is shown to outperform PSA-CMA-ESwRS, while others do not. This shows that despite BIPOP-aCMA-STEP being shown as the superior algorithm in the ECDF graphs, there are actually cases where some other, apparently worse algorithm, can still perform better. The next subsection will examine how exactly this can happen, and provide more context to the results obtained from the ECDF approach.

Finally, some algorithms did not use the maximum allowed $10^6 D$ problem evaluations. They instead used fewer problem evaluations, with most of these algorithms using $10^5 D$ evaluations. Because of this, we also conducted a separate statistical analysis where we selected results from only the first $10^5 D$ problem evaluations. Table 3.20 shows the results of this analysis and is structured in the same way as Table 3.14. As expected, algorithms that did not provide the results for the maximum number of evaluations perform better here. However, BIPOP-aCMA-STEP still performs very well, and shows overall the best performance for all dimensions except $D = 40$. In 40-dimensional problems, the algorithms CMA-CSA (Atamna, 2015) and CMA-TPA (Atamna, 2015) tie with NBIPOPaCMA, since in the pairwise comparison, these two algorithms did not outperform BIPOP-aCMA-STEP.

3.3.2.3 ECDF Versus Deep Statistical Approaches

In order to understand how the ECDF analysis produces the results that it does, we calculated the data shown in Tables 3.21 and 3.22. Table 3.21 shows the number of successful runs each algorithm achieved on a given problem. This is the same data that is available in the ECDF tables published by the competition organizers. Here, we can see that

Table 3.20: Results of the statistical analysis using results data limited to $10^5 D$ problem evaluations. Each column shows the number of algorithms a given algorithm performed better than. Algorithms marked with an asterisk (*) are those that did not provide data for the maximum $10^6 D$.

Algorithm	$D = 5$	$D = 10$	$D = 20$	$D = 40$
IPOP-SEP-CMA-ES*	0	7	12	-
BIPOP-CMA-ES*	7	8	9	9
iAMALGAM*	0	4	8	7
NELDERDOERR	0	0	0	-
AMALGAM*	0	4	6	4
VNS	0	1	3	-
DE-F-AUC*	3	0	1	1
DEuniform*	1	0	0	0
IPOP-ACTCMA-ES*	1	6	10	-
IPOP-CMA-ES*	0	6	8	-
MOS*	4	11	11	6
JADE*	0	4	1	3
JADEb*	0	0	0	1
JADEctpb*	0	4	1	3
NBIPOPaCMA	4	8	11	10
NIPOPaCMA	0	8	11	9
BIPOP-aCMA-STEP	13	14	20	8
CMA-CSA*	0	8	14	10
CMA-MSR*	0	7	9	9
CMA-TPA*	1	6	11	10
KL-BIPOP-CMA-ES	0	5	8	-
KL-IPOP-CMA-ES	0	5	10	-
KL-Restart-CMA-ES	6	6	9	-
PSA-CMA-ES	1	3	2	2
PSA-CMA-ESwRS	0	4	7	7

BIPOP-aCMA-STEP outperformed the other algorithms in problems 3 and 4. It reached the target fitness value in all 15 runs, while the others did not reach it. We interpret this to be the cause of the large gap between BIPOP-aCMA-STEP and the other algorithms in the ECDF graphs since those graphs look at the percentage of all successful runs.

Table 3.21: The number of runs that successfully solve a given benchmark problem, used by the ECDF analysis. We also provide the total sum of all solved runs, as well as the ratio of successful runs compared to all runs, rounded to two decimals.

Problem	**12-NB**	**12-NI**	**13-BI**	**18-PS**
1	15	15	15	15
2	15	15	15	15
3	0	0	15	0
4	0	0	15	0
5	15	15	15	15
6	15	15	15	15
7	15	15	15	15
8	15	15	15	15
9	15	15	15	15
10	15	15	15	15
11	15	15	15	15
12	15	15	15	15
13	15	15	15	15
14	15	15	15	15
15	15	15	15	15
16	15	15	15	15
17	15	15	15	15
18	15	15	15	15
19	9	15	8	11
20	0	0	0	0
21	15	12	14	15
22	12	0	4	2
23	15	15	15	0
24	2	4	1	1
Sum	293	286	312	269
Ratio	0.81	0.79	0.87	0.77

To illustrate the deep statistical approach, we present Table 3.22. Here, we show the relations between the rankings obtained by the DSC ranking scheme on each problem separately that are used by the deep statistical comparison to determine if one algorithm is significantly better than the other. In the comparison, each pair of algorithms (NIPOPaCMA, PSA-CMA-ESwRS), (NBIPOPaCMA, PSA-CMA-ESwRS), and (BIPOP-aCMA-STEP, PSA-CMA-ESwRS) is ranked against one another based on the statistical distribution of the results of each problem. For each problem, the + sign means that the first algorithm in the comparison (NIPOPaCMA, NBIPOPaCMA, or BIPOP-aCMA-STEP) performed better than PSA-CMA-ESwRS (i.e., both algorithms have different distributions of the results, but the first one has a smaller mean error), the = sign means that it performed equally well (i.e., both algorithms have the same distribution of the results), and the - sign means that it performed worse (i.e., both algorithms have different

distributions of the results, but the PSA-CMA-ESwRS has a smaller mean error).

Table 3.22: Ranks computed by Deep Statistical Comparison that are used for the statistical comparison for the algorithms NBIPOPaCMA, NIPOPaCMA, and BIPOP-aCMA-STEP

Problem	**12-NB**	**12-NI**	**13-BI**
1	=	=	=
2	=	=	=
3	+	+	+
4	+	+	+
5	=	=	=
6	=	=	=
7	=	=	=
8	=	=	=
9	=	=	=
10	=	=	=
11	=	=	=
12	=	=	=
13	=	=	=
14	=	=	=
15	=	=	=
16	=	=	=
17	=	=	=
18	=	=	=
19	=	+	=
20	+	+	=
21	=	−	=
22	+	=	=
23	+	+	+
24	=	=	−

Here, we can see that the comparison produces different results than the ECDF one. For example, BIPOP-aCMA-STEP, NIPOPaCMA, and NBIPOPaCMA, all outperform PSA-CMA-ESwRS on problems 3 and 4. In the ECDF evaluation, BIPOP-aCMA-STEP solved all 15 runs of these two problems, while the other algorithms solved zero. However, when taking into account the actual error values, we can see that even though NIPOPaCMA and NBIPOPaCMA fail to solve these problems, they still solve them statistically significantly better than PSA-CMA-ESwRS. A similar result occurs in problem 20. All algorithms fail to solve it in any of their runs, which contributes to their performance being closer in the ECDF evaluation. However, the deep statistical analysis reveals the differences between these algorithms. Problem 24 shows an even bigger difference. This difference in performance between the DSC and the ECDF approaches is consistent with prior work on this topic (Eftimov & Korošec, 2018).

Both the ECDF and the deep statistical analysis also revealed that a large number of test problems seem easily solvable. Problems 5 to 18 were all solved correctly in all runs by all of the four examined algorithms, as were problems 1 and 2. The deep statistical analysis also considered all algorithms equivalent on these problems. Together, these 15 problems represent more than half of the overall testing set. As these problems were all solved

correctly, they do not give as much insight into the performance of the three algorithms that we examined. However, the ECDF results evaluate not just whether or not a solution was reached, but also the speed at which the solution was found, which means that these problems still provide some information.

If we go through all benchmark problems, we can see that NBIPOPaCMA has statistically better performance against PSA-CMA-ESwRS on five problems, and on the other 19 problems, there is no statistical significance. The NIPOPaCMA has statistically better performance against PSA-CMA-ESwRS on five problems, on one problem it has statistically worst performance, and on the other 18 problems, there is no statistical significance between them. In the case of BIPOP-aCMA-STEP, it has statistically significantly better performance than PSA-CMA-ESeRS on three problems, on one problem it has statistically worst performance, and on the other 20 problems, there is no statistical significance between them. However, the differences that exist between the rankings obtained for each problem and the number of problems included in the benchmark data set influence the Wilcoxon test statistic. Even if the differences are only on a small number of problems, this can influence the test statistic that lead to the p-value. This is the reason why NBIPOPaCMA has statistically better performance than PSA-CMA-ESwRS, and there is no statistical significance between (NIPOPaCMA, PSA-CMA-ESwRS) and (BIPOP-aCMA-STEP, PSA-CMA-ESwRS) when performing multiple-problem analysis.

3.3.3 Cross-Benchmark Analysis

Finally, we present the results of the cross-benchmark analysis. The results are presented in three parts. In the first part, we show how the CEC algorithms performed when solving the problems from the BBOB benchmark set. In the second part, we present the performance of the BBOB algorithms on the CEC benchmark set. In the third part, we analyze how both the CEC and the BBOB algorithms perform when using different problem instances of the BBOB benchmarks, in order to verify our previous claims that the different instances do not greatly affect algorithm performance.

In order to perform this comparison, we chose the algorithms UMOEA and HSES from the CEC competitions, as these two algorithms were previously identified as the best performing algorithms on the CEC problem set. From the BBOB workshops, we chose the algorithms BIPOP-aCMA-STEP and JADE. BIPOP-aCMA-STEP was the only algorithm out of the ones compared in Tables 3.16, 3.17. 3.18, and 3.19 for which the source code was available on the workshop's website. In addition, JADE was selected as it showed good performance on lower dimensional BBOB problems, and because unlike most well-performing algorithms that have their source code available, it is not an algorithm based on CMA-ES. This means that including JADE lets us analyze the performance of two different types of algorithms.

We ran the CEC algorithms on the BBOB problems using their default set of parameters and using the BBOB instance 1-15. No modifications were made to the CEC algorithms. Similarly, the BBOB algorithms were run on the CEC problems with no changes to their parameters.

Figures 3.7-3.9 show the performance of the CEC algorithms on the BBOB problems on dimensions $D = 5$, $D = 10$, and $D = 20$, compared to the BBOB algorithm BIPOP-aCMA-STEP. The dimension of $D = 40$ was not used for this comparison, because the implementation of the HSES algorithm contained different logic for the cases where $D <= 30$ and where $D >= 50$, but no specific logic for the case where $D = 40$, and we did not want to modify its implementation. We can see that the CEC algorithms achieve a noticeably worse ECDF result. We can also observe that the two CEC algorithms maintain their relative performance based on dimension that was observed in the CEC analysis, with

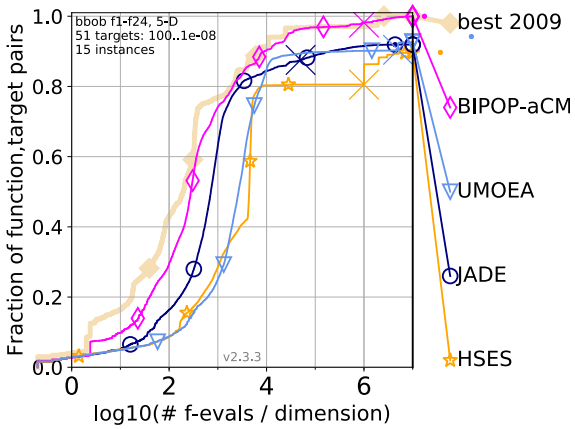

Figure 3.7: ECDF comparison of the CEC algorithms HSES and UMOEA executed on the BBOB problems, compared to the BBOB algorithms BIPOP-aCMA-STEP and JADE, using five-dimensional problems.

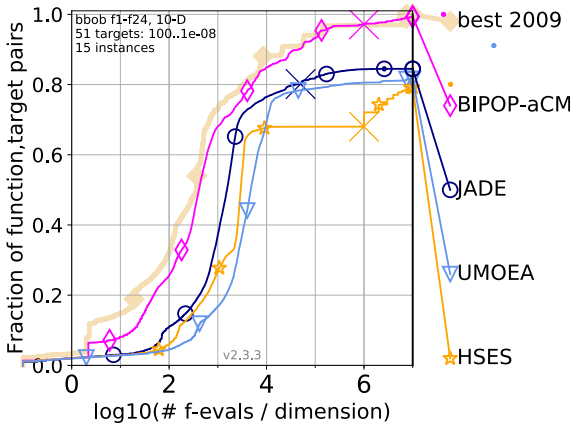

Figure 3.8: ECDF comparison of the CEC algorithms HSES and UMOEA executed on the BBOB problems, compared to the BBOB algorithms BIPOP-aCMA-STEP and JADE, using 10-dimensional problems.

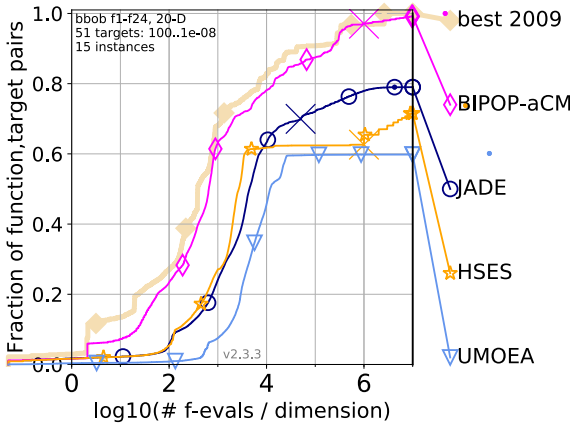

Figure 3.9: ECDF comparison of the CEC algorithms HSES and UMOEA executed on the BBOB problems, compared to the BBOB algorithm BIPOP-aCMA-STEP and JADE, using 20-dimensional problems.

UMOEA performing better than HSES in lower dimensions and worse in higher dimensions.

These results were expected, as the CEC algorithms were not designed to solve the BBOB benchmark problems. They were additionally not designed to take into account the BBOB experimental setup, for example its larger amount of available problem evaluations. Finally, the ECDF evaluation used by the BBOB workshops also measures the speed of convergence, rather than just the best-achieved result as is the case in the CEC competitions. Because of this, the CEC algorithms might not prioritize the speed of the convergence.

Table 3.23 shows the results of the deep statistical comparison of the BBOB algorithms BIPOP-aCMA-STEP and JADE when executed on the CEC 2017 problem set, compared with the CEC algorithms UMOEA and HSES over the dimensions of $D = 10$, $D = 30$, $D = 50$, and $D = 100$, with each algorithm executed in a total of 31 independent runs. We can see that the CEC algorithms generally outperform the BBOB algorithms. Specifically, UMOEA outperforms both BBOB algorithms in dimension $D = 10$, and HSES outperforms both algorithms in dimensions $D = 50$ and $D = 100$. The dimension $D = 30$ is the only dimension in which there is no statistically significant difference in performance between the BBOB algorithm BIPOP-aCMA-STEP and the two CEC algorithms.

Table 3.23: The p-values of a Wilcoxon test comparing the 2017 CEC algorithms (rows) to the BBOB algorithms (columns), executed on the CEC problems on problem dimensions $D = 10$, $D = 30$, $D = 50$, and $D = 100$. A value lower than 0.05 means that the CEC algorithm in the row outperformed the BBOB algorithm in the column.

(a) $D = 10$

	BIPOP	JADE
UMOEA	0.02	0.00
HSES	0.19	0.15

(b) $D = 30$

	BIPOP	JADE
UMOEA	0.10	0.00
HSES	0.73	0.01

(c) $D = 50$

	BIPOP	JADE
UMOEA	0.15	0.00
HSES	0.03	0.01

(d) $D = 100$

	BIPOP	JADE
UMOEA	0.84	0.00
HSES	0.00	0.00

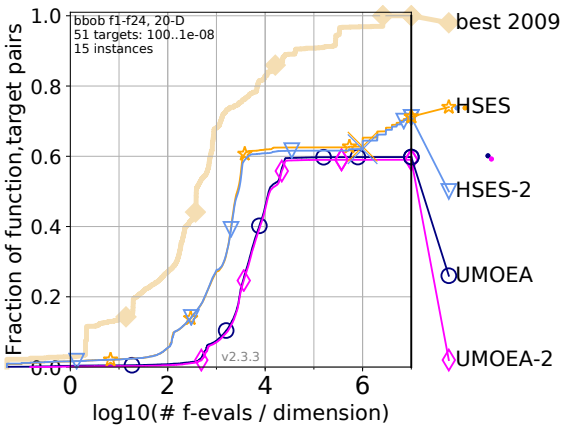

Figure 3.10: ECDF comparison of the CEC algorithms HSES and UMOEA executed on the BBOB problems using two different sets of instances, using 20-dimensional problems.

In order to verify our claim that different BBOB instances do not affect algorithm performance, we evaluated the CEC algorithms HSES and UMOEA on two different groups of BBOB instances, the first one consisting of instances 1-15 and the second consisting of instances 81-95, and using the dimensions of $D = 5$, $D = 10$, and $D = 20$. Figure 3.10 shows the results comparing the performance of both the BBOB and the CEC algorithms on two sets of BBOB instances, using the dimension of $D = 20$. Because the other dimensions of $D = 5$ and $D = 10$ achieved very similar results, we do not include their plots in this thesis. The results that are marked with only the name of each algorithm were evaluated on the BBOB instances 1 to 15, which were used in the 2009 edition of the workshops. The results marked with the name of each algorithm followed by the number 2 were evaluated on the BBOB instances 81 to 95, which were not used in any of the BBOB workshops up to 2018. We can see that the performance of algorithms is practically identical between the two sets. While this is not a thorough statistical analysis, the findings of this experiment show practical support for our theoretical assertion that different instances should not greatly affect algorithm performance, which allows us to compare the algorithms from the different years of the workshops.

3.4 Discussion and Conclusion

In this chapter, we have presented an analysis of the performance space of optimization by analyzing the results of the two most commonly used single objective optimization benchmarking events: the CEC competitions and the BBOB workshops. Our analysis of both of these revealed that there is a lack of improvement in overall algorithm performance from 2013 to 2018. In addition, in both events, we have observed that some algorithms that perform poorly overall can still perform very well on specific sets of problems.

The examination of the results of the CEC competitions shows a lack of improvement in overall algorithm performance throughout the years of the competitions, as neither EBO (from 2017) nor HSES (from 2018) managed to consistently outperform older algorithms. On the other hand, the 2013 algorithm NBIPOPaCMA showed good results in the 2017 benchmark set, even when using its default parameters. This means that its performance could potentially be even better on this benchmark set if the parameters were tuned to its problems.

When performing literature review, we noticed that most of the benchmark algorithms would compare their performance only to the best performing algorithms of the previous year of the competition. Such a comparison seems reasonable under the assumption that every year the new algorithms will improve on previous algorithms with respect to all problems and dimensions. However, the results of our analysis show that this is not the case. This means that for a more thorough analysis, the algorithms should also be compared with algorithms from older years of the competition.

Parameter tuning done by the algorithm authors may have had some effect on the overall results. LSHADE_EpSin and UMOEA performed the best on the 2014 benchmark set, outperforming all other algorithms on this set. However, while they did perform relatively worse on other benchmark sets, they still showed good overall performance. Over all benchmark problems, HSES performed well on high-dimensional problems, while UMOEA was the best at low-dimensional problems. Interestingly, the 2013 algorithm NBIPOPaCMA had the worst relative performance on the 2013 benchmark set but performed better on newer benchmark sets (particularly 2017) without any additional tuning.

The analysis of the BBOB workshops shows several important points. First, as with the CEC competitions, we see a lack of improvement in overall algorithm performance throughout the years. Specifically, the BIPOP-aCMA-STEP algorithm, which was pre-

sented in 2013, shows the best results. Another important point is the high amount of problems that were easily solved by a large number of algorithms. This indicates that a large number of problems are too easy to help assess algorithm performance. However, we note that the BBOB workshops are designed to also evaluate convergence speed, rather than just the final result achieved, which differs from the CEC competitions and our own analysis. It also shows that looking only at overall benchmark performance is not enough for a detailed assessment of algorithm performance. For example, when examining performance on individual problems, the NIPOPaCMA algorithm did not achieve the best overall performance. However, it was still the best performing algorithm when solving problem 22.

The BBOB experiments show the difference between ECDF and deep statistical analysis of benchmark results. Both the deep statistical and the ECDF approaches showed BIPOPaCMA-STEP to perform well. But the ECDF evaluation showed it as being much better than the competition, while the deep statistical comparison found it to perform equally well as a large number of other algorithms. This shows that even when there is an ECDF difference between algorithms, it might not be statistically significant. However, the ECDF results to allow for an analysis of not just the final solution reached, but also of how fast the solutions were reached.

Finally, both the CEC and the BBOB analyses show a lack of improvement in overall algorithm performance. It also shows that the overreliance on benchmark tests as a method for evaluating algorithm performance might not produce algorithms that work any better than the competition overall, but instead algorithms that are better specialized to a particular benchmark problem set. We also note the difference in the best performing algorithms between the two benchmark problem sets. In the CEC competitions, the best performing algorithms overall were the LSHADE_EpSin, UMOEA, and HSES algorithms. In the BBOB workshops, the best performing algorithms overall were variants of the CMA-ES algorithm. This further illustrates that an algorithm that performs well on one set of problems will not necessarily perform well on a different set of problems. The cross-benchmark experiments that we performed support this conclusion, as the algorithms designed for one set of problems performed worse when evaluated on a different set of problems.

While optimization benchmarking events might not lead to an overall improvement in algorithm performance, the events still provide important information on how specific problems solve individual problems, and more importantly, which problems are solved well by which algorithms. This information is in a sense more important than the overall algorithm performance because it gives us a more detailed analysis of an algorithm's performance. Additionally, this information can be used for the task of algorithm selection. However, this data by itself only allows for algorithm selection over the problems that are a part of the benchmark problem set, as it does not provide us with relevant information on the performance of algorithms on problems outside of the benchmarks. In order to extend our knowledge of algorithm performance to some new, unknown problem, we first require some way of describing optimization problems, and a way to determine which of the existing benchmark problems the new problem is most similar to. We will tackle these issues in the next two chapters, first by exploring a method for describing optimization problems, and second by analyzing this method's applicability to the algorithm selection problem.

Chapter 4

Fitness Landscape Analysis

As has been shown in the previous chapter, the performance of individual optimization algorithms is heavily dependent on the specific problem that is being solved. Additionally, no single optimization algorithm is able to achieve optimal performance on every single problem. This shows that in order to truly understand the performance and behavior of optimization algorithms, it is important to also understand the problems that the algorithms are solving. In this chapter, we will analyze the drawbacks of one current state-of-the-art method developed for the purpose of explaining the problem space of numerical optimization. Specifically, we will examine the performance of exploratory landscape analysis (ELA). The core principle of ELA is to use a small set of problem samples collected from an optimization problem and transform these samples into numerical descriptors called landscape features which describe the properties of the optimization problem. Ideally, these landscape features should be correlated with algorithm performance. That is, an algorithm that solves a specific problem well should also perform well on problems with similar landscape features. This property is important from a practical point of view because it would allow us to use ELA for tasks such as algorithm selection or to predict the performance of an algorithm on a specific problem by using a machine learning model.

This relation between landscape features and algorithm performance has been shown in practice. For example, ELA has been used to successfully predict the performance of algorithms on the BBOB problem set (Kerschke & Trautmann, 2019a). However, this evaluation has been relatively limited, and in particular, focused primarily on the problems from the BBOB benchmark.

In this chapter, we examine how landscape features behave when the samples used to calculate them have been transformed by certain mathematical transformations, specifically those of shifting and scaling. Transforming the samples in such a way should be simple enough that we do not expect it to have a large effect on the performance of most state-of-the-art optimization algorithms (i.e., we expect that the algorithms are invariant to these transformations).

In the first part of this chapter, we conduct a comprehensive analysis of the effect of problem transformations on landscape features. We determine which landscape features are affected by shifting and scaling, and which are invariant to the two transformations, and then examine how factors such as problem dimension, sample size, and the choice of the problem set affect the invariance of landscape features. In the second part of this chapter, we experimentally verify the effect of problem transformations by performing a complementarity analysis of problems from two different benchmark sets using landscape feature representations. These two sets share some benchmark problems, however, they apply different transformations to these problems. We show that removing the subset of features that are affected by the transformations of shifting and scaling greatly improves

the results of the complementarity analysis. The source code used for the visualization experiments in this Chapter is available at https://github.com/UrbanSkv/ela-visualization.

4.1 Effect of Shifting and Scaling on Landscape Features

We first explore the invariance of ELA features to the transformations of shifting and scaling. In this Chapter, we perform these transformations of vectors of samples $[x_1, x_2, ..., x_n]$ and $[y_1, y_2, ...y_n]$. The shifting transformation transforms these samples into $[x_1 + c_1, x_2 + c_2, ..., x_n + c_n]$ and $[y_1 + c_1, y_2 + c_2, ...y_n + c_n]$, while the scaling transformation transforms the samples into $[x_1c_1, x_2c_2, ..., x_nc_n]$ and $[y_1c_1, y_2c_2, ...y_nc_n]$, where $c_1, ..., c_n$ are randomly chosen real numbers. We will perform two different kinds of experiments, one where $c_1, ..., c_n$ all equal to a single randomly chosen number, and one where $c_1, ..., c_n$ each equal a different randomly chosen number.

We also note that we perform these transformations directly on the samples rather than directly on the problem, which is an important distinction due to using a limited sampling range. Transforming the samples directly means that, no matter the values of $c_1, ..., c_n$, the problem will always be sampled in its original sampling area.

We will refer to the landscape features that are not affected by these transformations as invariant features. In order to create a more robust analysis, we also analyze how the choice of various factors affects the invariance of landscape features. Specifically, we examine the factors of problem dimension, sample size, sampling strategy, and problem set on the invariance of landscape features.

4.1.1 Methodology

In order to determine which landscape features are invariant to the transformations of shifting and scaling, we devise the following methodology:

1. Using a set of optimization benchmark problems, create a set of problem samples S with a sample size of n_{samp} using a sampling method m_{samp} for every problem in the benchmark set.

2. Apply a shift or scale transformation to S to obtain a transformed sample set $\tilde{S}$ by either adding or subtracting a constant s_{const} to every dimension of every sample for shifting, by multiplying every dimension of every sample by a constant r_{const} for scaling, or by performing both shifting and scaling together. We call this the constant value scenario.

3. Alternatively, apply the shifting and scaling transformations by adding or subtracting a different random constant $s_{rand} \in (0, s_{const}]$ to every dimension separately, multiplying every dimension by a different random constant $r_{rand} \in (0, r_{const}]$, or by performing both of these operations together. We call this the random value scenario.

4. Calculate landscape features for the sample sets S and $\tilde{S}$ to obtain the sets of landscape features S_f and $\tilde{S}_f$.

5. Use a paired Wilcoxon signed-rank test to determine which landscape features contain statistically significant differences between the sets S_f and $\tilde{S}_f$. A single Wilcoxon test is performed on all of the problems together, as a paired comparison between the original and the transformed sets.

6. Repeat the above experiments using different sampling methods, dimensions, sample sizes, and problem sets, and by using different values for constants s_{const} and r_{const} for

shifting and scaling. The Wilcoxon test is performed separately on every repetition, and separately on each shifting and scaling factor.

7. Create a final set of *invariant* landscape features which contains only the landscape features that do not show a statistically significant difference between S_f and $\tilde{S}_f$ for all of the performed Wilcoxon tests.

A flowchart of this process is provided in Figure 4.1.

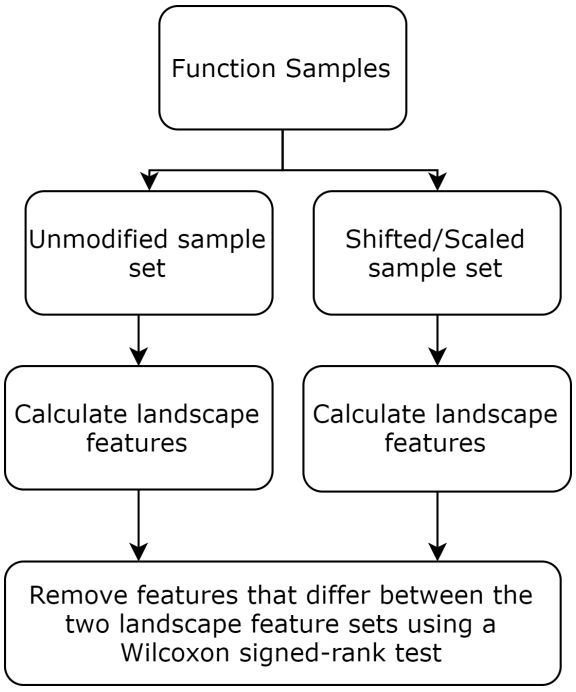

Figure 4.1: A visualization of the experimental process used to calculate the invariant features, from (Škvorc et al., 2020).

4.1.2 Experiments

We performed the above methodology in two different experiments. In the first, we determine which landscape features are invariant to the transformations of shifting and scaling under different choices of sampling strategies, and verify whether the choice of sampling strategy affects landscape feature invariance. In the second, we conduct a much more thorough examination of landscape feature invariance, taking into account sample size, problem dimension, and problem set.

4.1.2.1 Invariance With Regard to Sampling Strategies

The first experiment is motivated by prior work that has shown that the choice of sampling strategy has a large effect on the calculation landscape features (Renau et al., 2020). Because of this, when investigating the invariance of landscape features, we also wanted to consider multiple sampling strategies. We examined the following:

1. Uniform Random Sampling implemented using the MATLAB *rand* method.
2. Halton Sequence Sets (Halton, 1960) implemented using the MATLAB *haltonset* method.

3. Sobol Sampling (Saltelli et al., 2010) implemented using the MATLAB *sobolset* method.

4. Sobol Sampling with skips, leaps, and scrambling implemented using the MATLAB *sobolset* and *scramble* methods. The parameters used were 10^3 for the skip operation and 10^2 for the leap operation. The Matousek-Affine-Owen algorithm (Matoušek, 1998) was used for the scrambling operation.

5. Latin Hypercube Sampling (McKay et al., 2000), implemented using the MATLAB *lhsdesign* method.

In these experiments, we used the values of $s_{const} \in \{-0.1, -2, -5, -10, -50, -100, -1000, 0.1, 2, 5, 10, 50, 100, 1000\}$ and $r_{const} \in \{0.1, 2, 5, 10, 50, 100, 1000\}$. We chose these numbers to have a variety of both small and large values, and the number of parameters chosen was limited due to the computational complexity of the experiments. All possible combinations of s_{const} and r_{const} were used. Only the landscape features that were invariant under all of the combinations were considered to be invariant in the final results. For this experiment, we used only a single set of benchmark problems to calculate the invariant features, specifically the problem set used by the 2014 CEC competitions. We used a single sample size of $n_{samp} = 250D$, with a dimension of $D = 10$ in order to decrease computational complexity, and to isolate our experiments just to the sampling strategy.

For the calculation of landscape features, we used the open-source library flacco (Kerschke & Trautmann, 2019b). We did not use the full amount of landscape features provided and instead chose to limit ourselves to only a fraction of the landscape features that are provided by the library. We excluded the landscape features that are computationally expensive to compute, specifically on higher dimensions. We also excluded the features that require additional sampling to compute, as well as features that produced errors during computation. The full list of 66 landscape features used for our experiments is available in Table 4.1.

4.1.2.2 Invariance With regard to Sample Size, Dimension, and Problem Set

In the second experiment, we wanted to upgrade our previous analysis of the sampling strategy by including a much broader variety of factors that might affect the invariance of landscape features. In addition to the sampling strategy, we analyzed the following factors:

1. The problem set. We used the problem sets from the years 2013, 2014, 2015, and 2017 of the CEC competitions, as well as the first 15 instances of the BBOB benchmark set.

2. The problem dimension. We used the dimensions of $D = 10$ and $D = 30$.

3. The sample size. We used the sampling sizes of $n_{samp} = 50D$, $n_{samp} = 100D$, $n_{samp} = 200D$, and $n_{samp} = 250D$.

Figure 4.2 shows all of the factors that were considered in these two experiments. Because the first experiment already analyzed the effect of the sampling strategy, the second experiment only considered a landscape feature as invariant if it was found to be invariant for all five sampling strategies.

The second experiment was much more computationally expensive, as we wanted to examine every possible combination of sample size, strategy, dimension, and problem set. This analysis additionally included a higher dimension of $D = 30$, which further increased the computational complexity. Because of this, we only used the values $s_{const}, s_{rand} \in$

Table 4.1: The list of all 66 landscape features that were used for the invariance analysis, with the names that are used in the library flacco.

Feature Names 1–33	Feature Names 34–66
cm_angle.dist_ctr2best.sd	ela_meta.lin_simple.adj_r2
ela_meta.lin_w_interact.adj_r2	cm_angle.dist_ctr2worst.sd
cm_angle.dist_ctr2worst.mean	disp.diff_median_10
cm_angle.angle.sd	nbc.dist_ratio.coeff_var
cm_grad.mean	cm_angle.y_ratio_best2worst.mean
ela_distr.kurtosis	ic.h.max
cm_angle.y_ratio_best2worst.sd	ela_meta.lin_simple.coef.max_by_min
disp.ratio_median_05	cm_grad.sd
disp.ratio_mean_05	disp.diff_median_05
ela_distr.number_of_peaks	limo.ratio.mean
disp.diff_median_25	disp.ratio_median_02
disp.diff_median_02	ela_meta.lin_simple.intercept
limo.length.sd	cm_angle.dist_ctr2best.mean
ic.eps.max	limo.cor.reg
ela_meta.lin_simple.coef.max	disp.diff_mean_10
limo.cor.norm	nbc.nn_nb.mean_ratio
pca.expl_var_PC1.cor_init	limo.ratio.sd
cm_angle.angle.mean	pca.expl_var_PC1.cov_init
limo.sd_ratio.reg	limo.avg_length.norm
nbc.nb_fitness.cor	limo.sd_ratio.norm
ela_meta.quad_simple.cond	disp.diff_mean_25
limo.sd_mean.reg	disp.ratio_median_10
limo.avg_length.reg	limo.sd_mean.norm
ela_meta.lin_simple.coef.min	limo.length.mean
pca.expl_var.cov_x	disp.ratio_mean_10
ela_meta.quad_w_interact.adj_r2	pca.expl_var.cor_x
disp.ratio_mean_25	nbc.nn_nb.cor
pca.expl_var.cov_init	disp.diff_mean_02
nbc.nn_nb.sd_ratio	pca.expl_var.cor_init
disp.ratio_median_25	ic.m0
disp.ratio_mean_02	disp.diff_mean_05
pca.expl_var_PC1.cov_x	ela_distr.skewness
ela_meta.quad_simple.adj_r2	pca.expl_var_PC1.cor_x

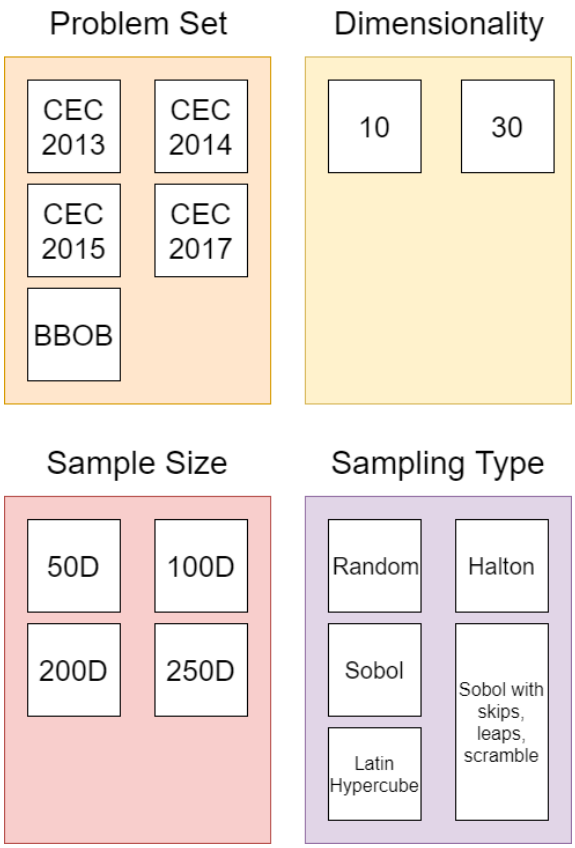

Figure 4.2: An overview of all of the problem sets, dimensions, sample sizes, and sampling types used in our experiments.

$\{-1000, 1000\}$ and $rconst, rrand \in \{0.1, 1000\}$, using the smallest and the largest values used in the first experiment. In addition, only the combined transformation of shifting and scaling was examined.

4.1.3 Results

In this section, we present the results of our analysis by showing which landscape features were found to be invariant to the transformations of shifting and scaling.

The results in this chapter are presented using upset diagrams. These diagrams were introduced by Alexander Lex et al. in (Lex et al., 2014) as alternatives to Venn diagrams for data with a large number of sets. In these plots, the size of the columns represents the number of landscape features invariant to the specific transformation based on the selected sampling method. The bottom part shows which methods belong to each column, with a single dot meaning that the column represents the landscape features invariant only under a specific sampling method, while multiple dots mean that these landscape features are invariant under all of the specified sampling methods. The left part of the diagram, labeled *Set Size*, represents the total number of invariant features for each individual sampling method.

4.1.3.1 Sampling Strategies

The results in this section present the results from both the constant value scenario (each dimension transformed using the same constant) and the random value scenario (each dimension transformed separately using a different random value).

Figure 4.3 shows the number of landscape features invariant to the transformation of shifting under the constant value scenario.

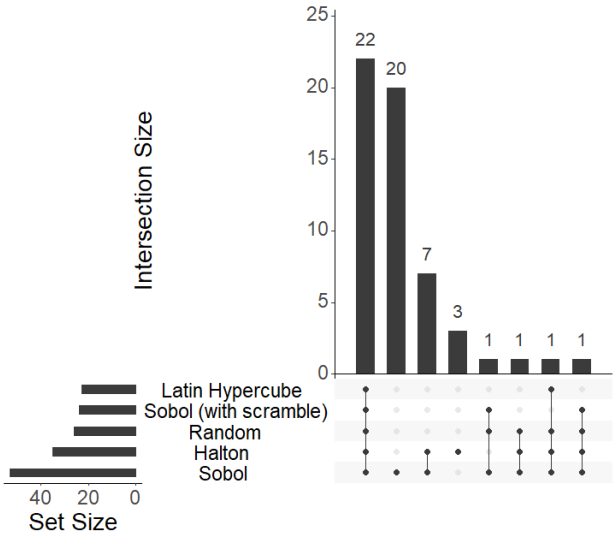

Figure 4.3: The number of landscape features invariant to shifting under different sampling strategies, with each dimension shifted by the same constant.

We can see that the selection of sampling method does affect the invariance of landscape features, as there exist landscape features that are invariant only under certain sampling methods. Out of a total of 56 features that are invariant under at least one transformation, only 22 are invariant under all transformations. Sobol sampling in particular is an outlier, with 20 features that are only invariant under this sampling.

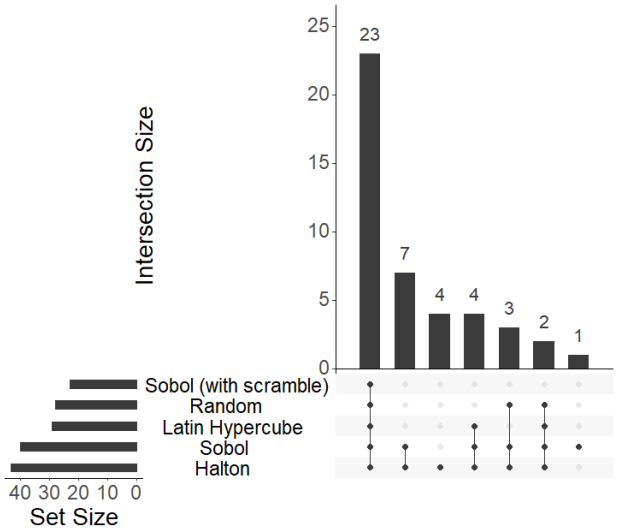

Figure 4.4: The number of landscape features invariant to scaling under different sampling strategies, with each dimension scaled by the same constant.

Figure 4.4 shows the same data for the transformation of scaling. Here, the results are lesser but still present. Out of a total of 44 landscape features that are invariant under at least one sampling method, 23 of them are invariant under all sampling methods. In addition, a total of 7 landscape features are only invariant under both Sobol and Halton sampling methods.

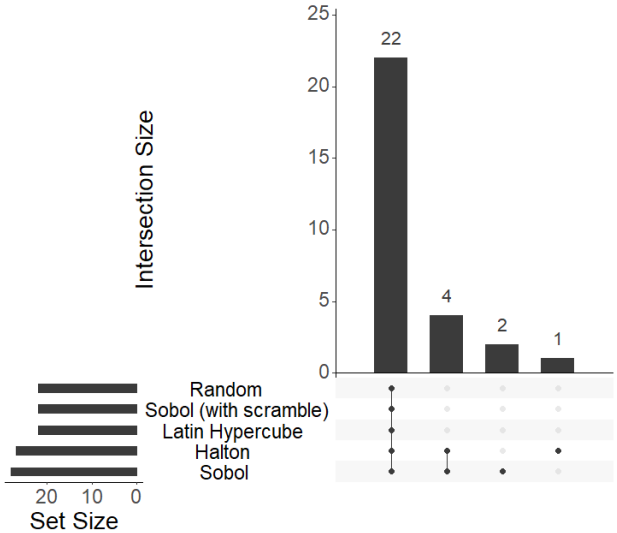

Figure 4.5: The number of landscape features invariant to the combined transformation of both shifting and scaling under different sampling strategies, with each dimension shifted by the same constant and scaled by the same constant.

Figure 4.5 shows the results for the combined transformation of both shifting and scaling. Here, the difference becomes even less drastic, and only a small number of features are invariant only under specific sampling methods. Note that in the combined case, when both the shifting and the scaling transformations apply a constant change to all dimensions, the constants used for scaling and shifting are not necessarily the same, as all possible

combinations of constants were used in this case.

Figure 4.6 shows the number of landscape features invariant to the transformation of shifting under the random value scenario. As this transformation more significantly changes the original sample set, we believed that it would have a greater effect on the invariance of landscape features. We can see that this is indeed the case, with only 15 landscape features now invariant under all sampling types, as opposed to 22 in the previous scenario.

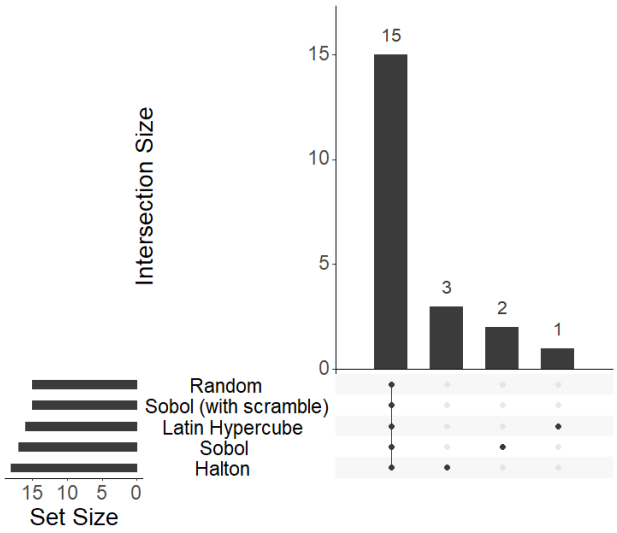

Figure 4.6: The number of landscape features invariant to shifting under different sampling strategies, with each dimension scaled separately with a different random value.

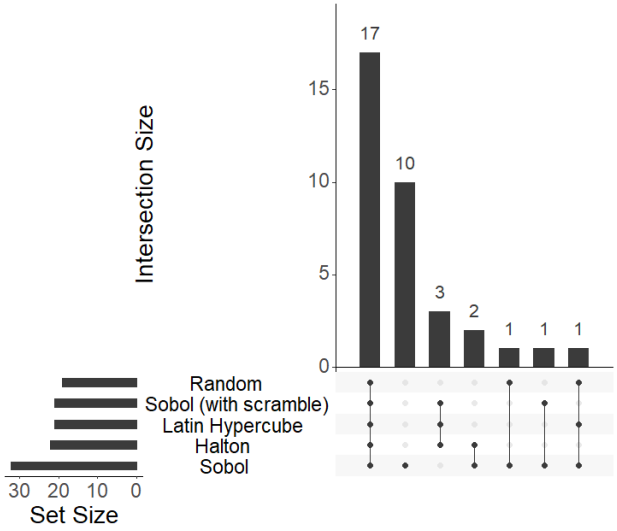

Figure 4.7: The number of landscape features invariant to scaling under different sampling strategies, with each dimension scaled separately with a different random value.

Figure 4.7 shows the same data for the transformation of scaling. Here, we see that Sobol sampling is now an outlier in the scaling transformation, rather than the shifting transformation seen in the constant value scenario.

Figure 4.8 shows the results for the combined transformation of both shifting and

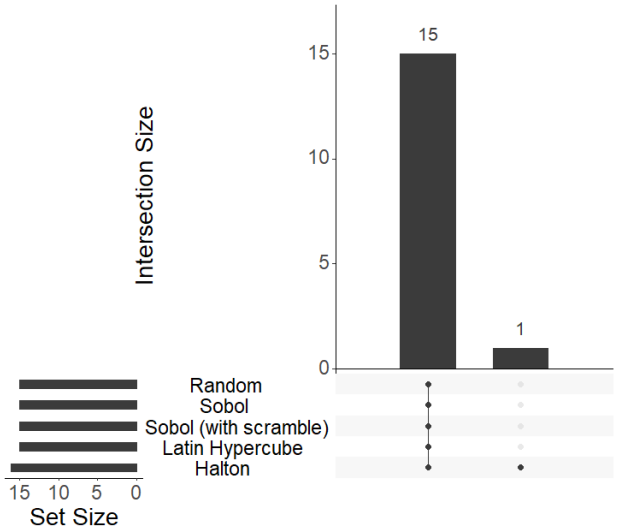

Figure 4.8: The number of landscape features invariant to the combined transformation of both shifting and scaling under different sampling strategies, with each dimension scaled and shifted separately with a different random value.

scaling. Here, we can see that the choice of the sampling strategy only has a very small effect, with only a single outlier feature that is not invariant under all sampling strategies.

Finally, we show which specific landscape features are invariant under all of the sampling methods. Tables 4.2, 4.3, and 4.4 show the results of our experiments, showing which features are invariant to the transformation of shifting, scaling, and a combined transformation of both shifting and scaling. In these tables, the landscape features marked in bold are invariant in both the constant and the random value scenarios, while the features not marked in bold are only invariant in the constant value scenario.

Out of the total of 66 landscape features that were used for the analysis, only 22 of them are invariant to the combined transformation of shifting and scaling when these transformations are performed using a constant shifting and scaling value across all dimensions.

When the transformations are performed using a random value separately for each dimension, this number drops to 15. This confirms that as far as feature invariance is concerned, using a different scaling and shifting factor for each dimension is a more difficult transformation. We can also see that these 15 features are a strict subset of the 22 features that are invariant to the transformations that use a single constant for all dimensions.

Figures 4.9 and 4.10 show the upset diagrams for these two cases. Notably, we can see that the set of features that are invariant to the combined transformation of shifting and scaling is the same as the set of features that are invariant just to shifting.

4.1.3.2 Sample Size, Problem Set, and Dimension

In this part of the results, we will focus on three different factors. First, we show how the invariance is affected by the choice of the sample size. Second, we show how it is affected by the choice of the problem set. Finally, we show how it is affected by the dimension of the problems. We do not present the effect of the sampling strategy individually, as this was already shown in the previous section. We also do not show the individual effects of each individual shifting and scaling constants s_{const} and r_{const}. Instead, the sampling strategy and the constants s_{const} and r_{const} are grouped together. We consider a feature to be invariant only if it is invariant under all sampling strategies and all constants s_{const}

Table 4.2: The landscape features that are invariant to the transformation of shifting under all five sampling methods. The features in bold are invariant in both the constant and random value scenarios, while the rest are only invariant in the constant value scenario.

Feature names
cm_angle.angle.sd
cm_angle.dist_ctr2best.sd
cm_angle.dist_ctr2worst.sd
cm_angle.y_ratio_best2worst.mean
cm_angle.y_ratio_best2worst.sd
cm_grad.sd
ela_distr.number_of_peaks
limo.cor.norm
limo.cor.reg
limo.length.sd
limo.ratio.sd
limo.sd_mean.norm
limo.sd_mean.reg
limo.sd_ratio.norm
limo.sd_ratio.reg
pca.expl_var.cor_x
pca.expl_var.cov_x
cm_grad.mean
ic.eps.max
ic.h.max
ic.m0
pca.expl_var.cor_init
pca.expl_var.cov_init

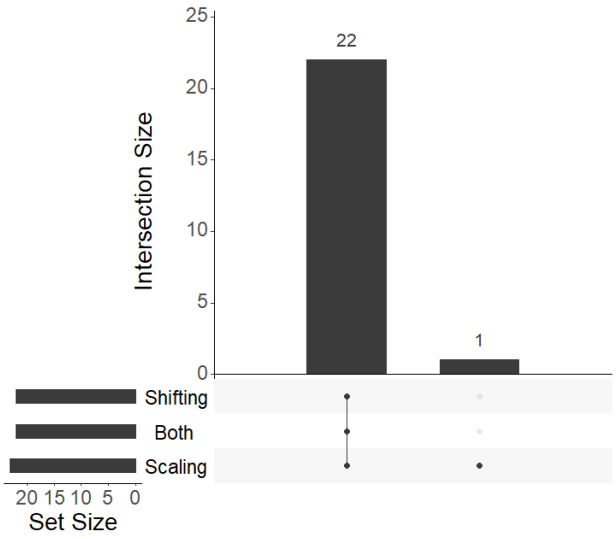

Figure 4.9: The number of landscape features invariant to the transformations of shifting, scaling, and the combined transformation under all sampling methods, with all dimensions shifted and scaled by the same constant.

Table 4.3: The landscape features that are invariant to the transformation of scaling under all five sampling methods. The features in bold are invariant in both the constant and random value scenarios, while the rest are only invariant in the constant value scenario.

Feature name
cm_angle.angle.sd
cm_angle.dist_ctr2best.sd
cm_angle.dist_ctr2worst.sd
cm_angle.y_ratio_best2worst.mean
cm_angle.y_ratio_best2worst.sd
cm_grad.sd
ela_distr.number_of_peaks
limo.cor.norm
limo.cor.reg
limo.length.sd
limo.ratio.sd
limo.sd_mean.norm
limo.sd_mean.reg
limo.sd_ratio.norm
limo.sd_ratio.reg
ic.eps.max
ic.h.max
ic.m0
pca.expl_var.cor_init
pca.expl_var.cor_x
pca.expl_var.cov_init
pca.expl_var.cov_x

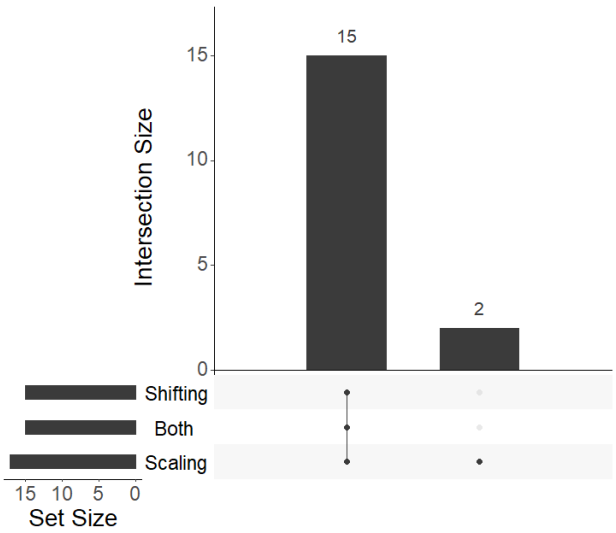

Figure 4.10: The number of landscape features invariant to the transformations of shifting, scaling, and the combined transformation under all sampling methods, with each dimension shifted and scaled separately with a different random value.

Table 4.4: The landscape features that are invariant to the combined transformation of shifting and scaling under all five sampling methods, with all dimensions shifted and scaled by a constant. The features in bold are invariant in both the constant and random value scenarios, while the rest are only invariant in the constant value scenario.

Feature name
cm_angle.angle.sd
cm_angle.dist_ctr2best.sd
cm_angle.dist_ctr2worst.sd
cm_angle.y_ratio_best2worst.mean
cm_angle.y_ratio_best2worst.sd
cm_grad.sd
ela_distr.number_of_peaks
limo.cor.norm
limo.cor.reg
limo.length.sd
limo.ratio.sd
limo.sd_mean.norm
limo.sd_mean.reg
limo.sd_ratio.norm
limo.sd_ratio.reg
ic.eps.max
ic.h.max
ic.m0
pca.expl_var.cor_init
pca.expl_var.cor_x
pca.expl_var.cov_init
pca.expl_var.cov_x

and r_{const}.

Figures 4.11 and 4.12 show the results of the effect of the sample size on the invariance of the landscape features to problem transformations. Figure 4.11 shows the results under a constant value scenario, while Figure 4.12 shows the results under a random value scenario.

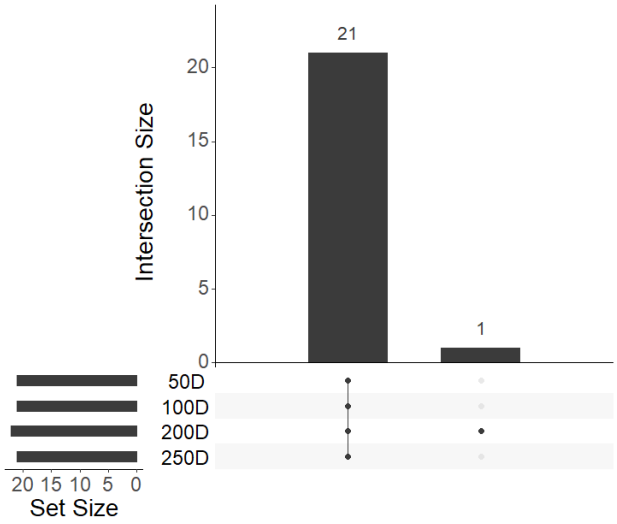

Figure 4.11: The effect of sample size on the invariance of landscape features, with each dimension shifted by the same constant and scaled by the same constant.

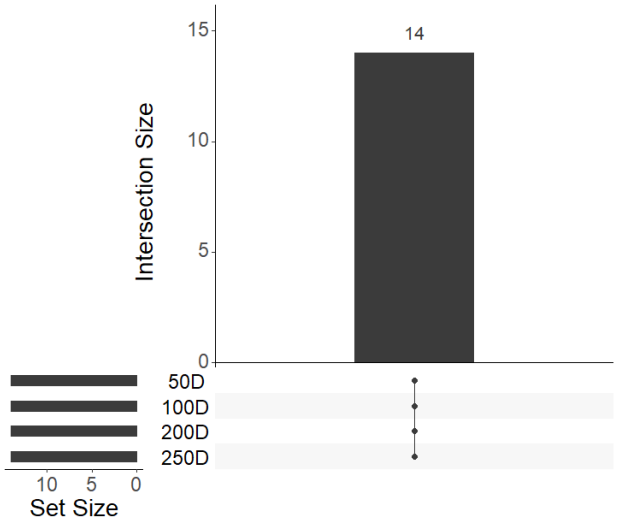

Figure 4.12: The effect of sample size on the invariance of landscape features, with each dimension shifted and scaled separately with a different random value.

We can see that the sample size does not have an effect on landscape feature invariance. In the random value scenario, 14 landscape features are invariant regardless of the sample size. In the constant value scenario, there are 21 features invariant regardless of the sample size, and a single outlier feature *ela_distr.number_of_peaks* that is invariant when using the sample size of $n_{samp} = 200$.

Upon examining the samples where the *ela_distr.number_of_peaks* feature is not invariant and performing additional experiments, we determined that this feature is sensitive

to very large outliers in the problem values y in the sample set. We verified this experimentally by creating an artificial set of samples x and values y by uniformly sampling values of x and y from the range of $(0, 10]$ using a dimension of $D = 10$ and sample size of $250D$. Then, a single sample value from y was replaced by an extremely large value $Y_{large} \in \{10^{10}, 10^{15}, 10^{16}, 10^{20}\}$.

Then, just as in the original experiment, we created an alternate set of samples by scaling the original x and y values by 0.1 and shifting them by 1000. The original and the shifted sets showed a difference in the *ela_distr.number_of_peaks* feature, but only when the outlier value Y_{large} was set to 10^{16} or 10^{20}. One possible explanation for this could be the numerical instabilities caused by the inexact nature of floating points used by most programming languages, including the R programming language.

Specifically, the R implementation of floating point numbers uses 53 bits for the significant precision ("Numerical Characteristics of the Machine," 2022), which gives it a precision of 15 to 17 significant digits ("IEEE Standard for Floating-Point Arithmetic," 2008). It is possible that, at large Y values, the relatively small shifting and scaling constants are not represented with enough precision, and might alter the relation between values due to rounding errors.

To further explain the non-invariance of some features, we calculated the amount of non-invariant sampling sets for every landscape feature that was classified as non-invariant. We found that the mean number of non-invariant sample sets is 8,298, and the median is 8,146.

Figure 4.13 shows the ratio of non-invariant sampling sets for each of the 52 non-invariant landscape features compared to the entire set of 16,000 sample sets. We can see that there are two non-invariant landscape features in every sampling set. These features are *cm_angle.dist_ctr2best.mean* and *cm_angle.dist_ctr2worst.mean*. After this, there are 11 landscape features that are not invariant in over 80% of all sample sets.

There are a total of five features that are not invariant in less than 10% of all sample sets. These features are *ela_meta.quad_simple.cond*, *ela_distr.skewness*, *limo.avg_length.norm*, *ela_distr.kurtosis*, and *ela_distr.number_of_peaks*. Of these, only one is not invariant in less than 160, or 1%, of all sampling sets, which is the *ela_distr.number_of_peaks* feature.

Specifically, it is invariant in only five sample sets. However, since we require that a feature be invariant in all sample sets in order to consider the feature as a whole invariant, even this small amount was enough to classify the feature as non-invariant.

Figures 4.14 and 4.15, show the results of the analysis grouped by a problem set. This grouping shows similar results to the grouping by sample size. The constant value scenario shows 21 features that are invariant regardless of the problem set, as well as two outliers. The first outlier, *nbc.nb_fitness.cor*, is not invariant in the problems from the years 2013, 2014, and 2017 of the CEC competitions. The second outlier is *ela_distr.number_of_peaks*, which was previously elaborated upon. The random value scenario shows a similar situation. However, the additional outlier in this case is *pca.expl_var.cov_init*.

As with the sample size grouping, we can see one outlier feature in both the constant and random value scenarios. In this case, the outlier is different depending on the scenario. The outliers are *nbc.nb_fitness.cor* in the constant value scenario and *pca.expl_var.cov_init* and *ela_distr.number_of_peaks* in the random value scenario. From this, we can see that the problem set for the most part does not affect landscape feature invariance. We note that these results are different from our previous research (Škvorc et al., 2022), which showed a larger number of outliers in the 2013 problem set that we could not explain at the time. After a thorough analysis, we concluded that these outliers were caused by the fact that in the implementation of the function responsible for calculating the landscape

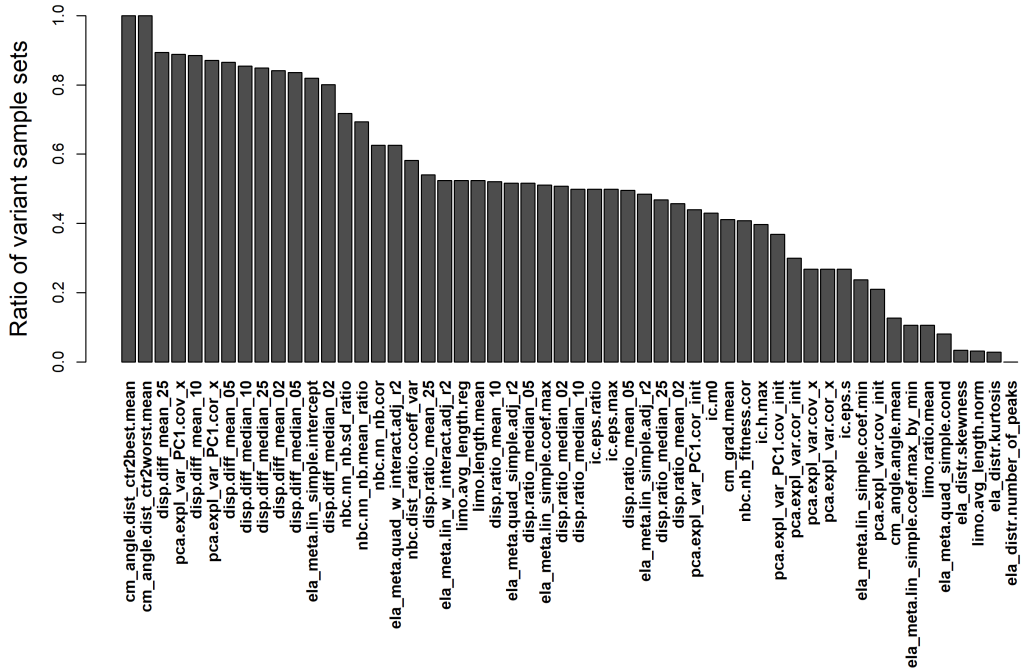

Figure 4.13: The ratio of non-invariant sampling sets for each landscape feature, out of a total of 16,000 sample sets.

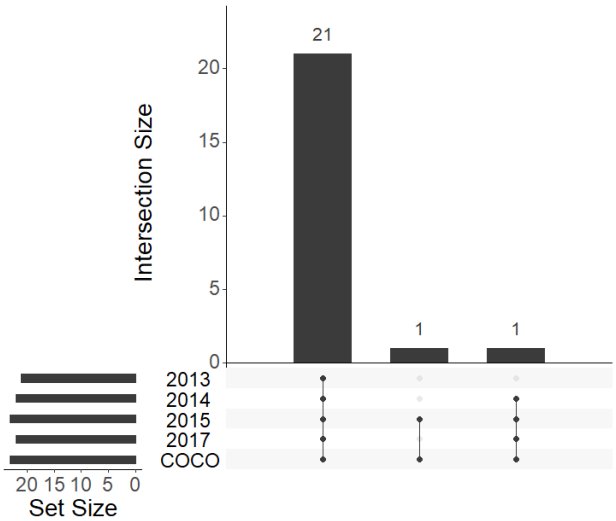

Figure 4.14: The effect of the problem set on the invariance of landscape features, with each dimension shifted by the same constant and scaled by the same constant.

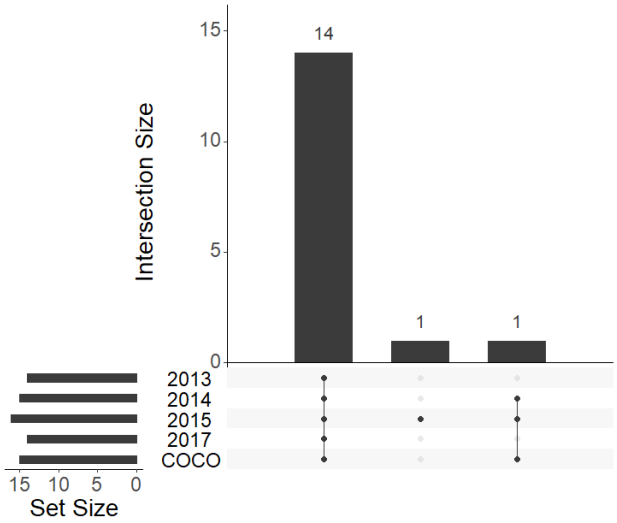

Figure 4.15: The effect of the problem set on the invariance of landscape features, with each dimension shifted and scaled separately with a different random value.

features, the random seed was set to a fixed value only at the start of the function. In most cases, this would ensure that the original and the transformed samples would use the same random seed in order to calculate the landscape features. However, if an error occurred during the feature calculation process, the feature set that produced the error would be skipped and its features set to a NA value. In this scenario, the fact that the feature was skipped resulted in the random seed no longer being consistent for the remaining features. Because the 2013 problem set contained one problem which would often produce errors when calculating landscape features, this resulted in outliers in the 2013 problem set, specifically among features that are sensitive to the choice of the random seed. This issue has been fixed, and all of the results in this thesis now set the random seed before each individual feature is calculated, meaning that an error in the calculation of one set of features no longer has an effect on other features.

Figures 4.16 and 4.17 show the effect of the dimension on the invariance of the landscape features. Here, the results are similar to the sample size analysis. The dimension of the data has no effect on the invariance, except for a single outlier in the constant value scenario, which is once again *ela_distr.number_of_peaks*.

Figure 4.18 shows the number of features that are invariant when considering every problem set, dimension, and sample size. We can see that under the constant scenario, there are a total of 21 invariant features, while under the random scenario there are a total of 14. These features are the same ones that were found to be invariant in (Škvorc et al., 2021), with the exception of the *ela_distr.number_of_peaks*. This analysis shows that introducing new sets of problems, increasing the dimension, or reducing the sample size does not have an effect on landscape feature invariance.

4.2 Complementarity Analysis

In this section, we verify that the experimental findings of the previous section also hold in practice by more closely examining whether an ELA-based problem representation can show the complementarity of two different benchmark problem sets. Specifically, we will use the BBOB problem set and the problem sets from two years of the CEC competi-

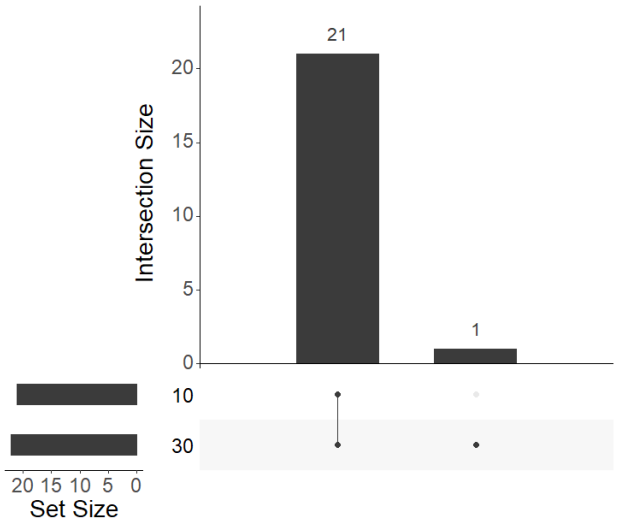

Figure 4.16: The effect of the dimension on the invariance of landscape features, with each dimension shifted by the same constant and scaled by the same constant.

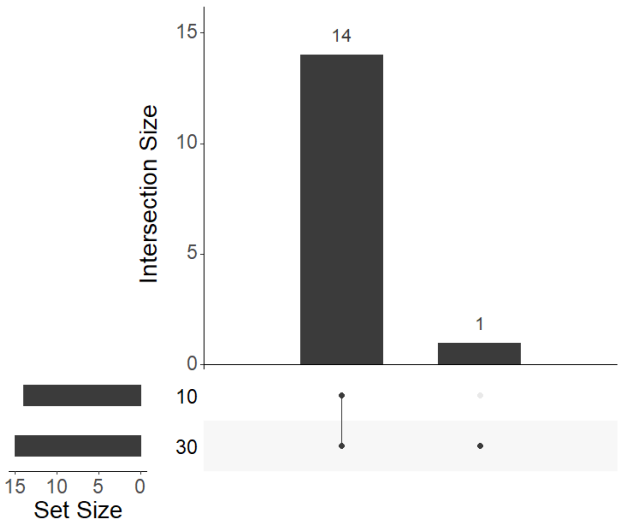

Figure 4.17: The effect of the dimension on the invariance of landscape features, using the random value scenario.

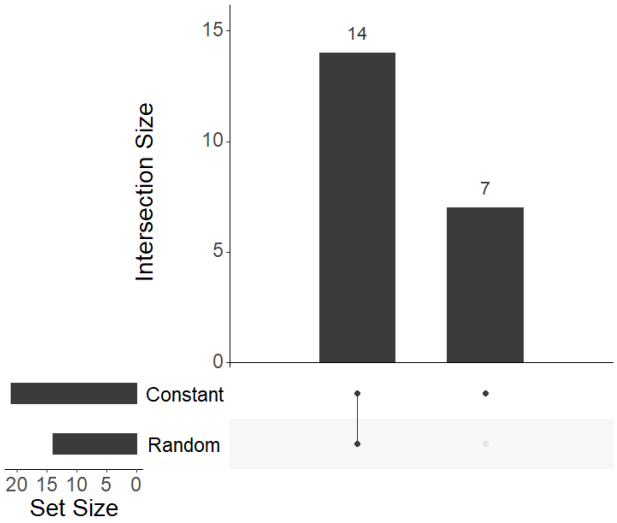

Figure 4.18: The combined effect of sample size, problems set, and dimension on the invariance of landscape features, using both the constant value and the random value scenarios.

tions. The complementarity of the problem sets will be evaluated by visualizing the ELA representation and examining if the problems from the two sets that should be similar to one another based on their problem definitions are also similar in terms of their ELA representation. This is similar to ELA-based visualization procedures used by prior work such as (Muñoz & Smith-Miles, 2017; Oliveira et al., 2018; K. Smith-Miles et al., 2014; K. Smith-Miles & Bowly, 2015).

We perform two different complementarity experiments. In the first, we compare the problems from the 2014 and 2015 CEC competitions. These problem sets come from the same benchmark, and share a number of problems that should be considered similar by the ELA-based representation. However, the two problem sets apply different transformations to the base problems, including shifting and scaling. We will show that if these transformations are not accounted for, the ELA-based representation does not properly represent the similarities between the problems.

In the second experiment, we compare the problems from the 2014 CEC competition and the BBOB problem set. Here, the two problem sets come from entirely different benchmarks, which we believe will introduce additional difficulty for the ELA-based representation. However, the two sets still share a number of problems, and these shared problems should be reflected in the results of our experiments.

4.2.1 Experimental Setup

For the CEC competitions, we could have used the entire dataset of all problems from all of the years of the competition. This would have provided us with a total of 102 problems. In order to ease the computational complexity of the complementarity analysis, we chose to focus mainly on two specific years of the benchmark (2014 and 2015), which contain a total of 45 problems. For the BBOB workshops, we used all 24 problems.

For both CEC and BBOB problems, we used the dimensions of $D = 2$ and $D = 10$. Using two-dimensional problems allows us to visually plot the problems, which gives us a much greater ability to explain and understand the results of our complementarity experiments. However, several problems in the CEC benchmarks are not defined for two

dimensions. 10-dimensional problems were chosen because the actual CEC competitions use dimensions of $D = 10$, $D = 30$, $D = 50$, and $D = 100$, while the BBOB workshops use the dimensions of $D = 2$, $D = 3$, $D = 5$, $D = 10$, $D = 20$, and $D = 40$, making 10 the only common dimension. We deliberately wanted both problem sets to have the same dimension to make the comparison as fair as possible.

In order to calculate the landscape features, the flacco library was used as in the experiments in the previous section. We used the landscape features from the following feature sets: *cm_grad, ela_conv, ela_distr, ela_level, ela_local, ela_meta, ic, disp, limo, nbc*, and *pca*. We used a total of 101 features, which is larger than the set in the previous experiments. This was chosen as the performance constraints in these experiments were lesser than in the previous experiments, as fewer factors had to be considered, as well as because the methodology used in this chapter resulted in fewer landscape features that produced errors during execution.

In order to calculate the features, we used improved hypercube sampling (Beachkofski & Grandhi, 2002), which reduces the variance of the samples, but is much more computationally expensive, which is why it was not evaluated in the previous section, which featured a much larger amount of computations than the experiments in this section. We used several different sample sizes in order to examine how sample size affects the results. We initially used a sample size of $n_{samp} = 50D$, based on a similar article by Kerschke and Trautmann (Kerschke & Trautmann, 2019a), where this sample size was used. However, as explained in Section 4.2.2, we also experimented with additional sample sizes. The samples chosen were from the range of $[-100, 100]$, which corresponds to the range used by the CEC competitions, and is larger than the $[-5, 5]$ range of the BBOB workshops.

We used the same method for eliminating invariant features as in the previous experiments. The invariant features were calculated using only the problems in the 2014 problem set. This was done to simulate a scenario where an experiment examines two sets of problems: a set of known problems and landscape features (in this case the 2014 CEC set), and a new set of unknown problems (in our scenario either the 2015 CEC or the BBOB set). In order to not overfit the data, only the known problem set is used for the invariance calculation.

After the non-invariant features are removed, we also removed all features with constant values. Finally, we used principal component analysis (PCA) (Wold et al., 1987), and selected the components that together explained 75% of the dataset variance. This threshold value was chosen experimentally as it produced the best visualizations.

Figure 4.19 shows a visual representation of the entire process.

4.2.2 Results

In this section, we present the results of the complementarity analysis in three subsections. First, we briefly show the features used for the analysis after removing the non-invariant and correlated features and after performing PCA.

Second, we show the results of our analysis on a simple test case by comparing benchmark problems from different years of the CEC competitions. The goal of this subsection is to evaluate whether the ELA-based representation can correctly identify similar problems from the two CEC sets, which should be relatively similar to one another.

Finally, we expand this analysis to comparisons between two different benchmark settings: the CEC competitions and the BBOB workshops. The goal of this subsection is to test if the ELA-based representation can be used to correctly compare problems from benchmark sets with a relatively small amount of similarities.

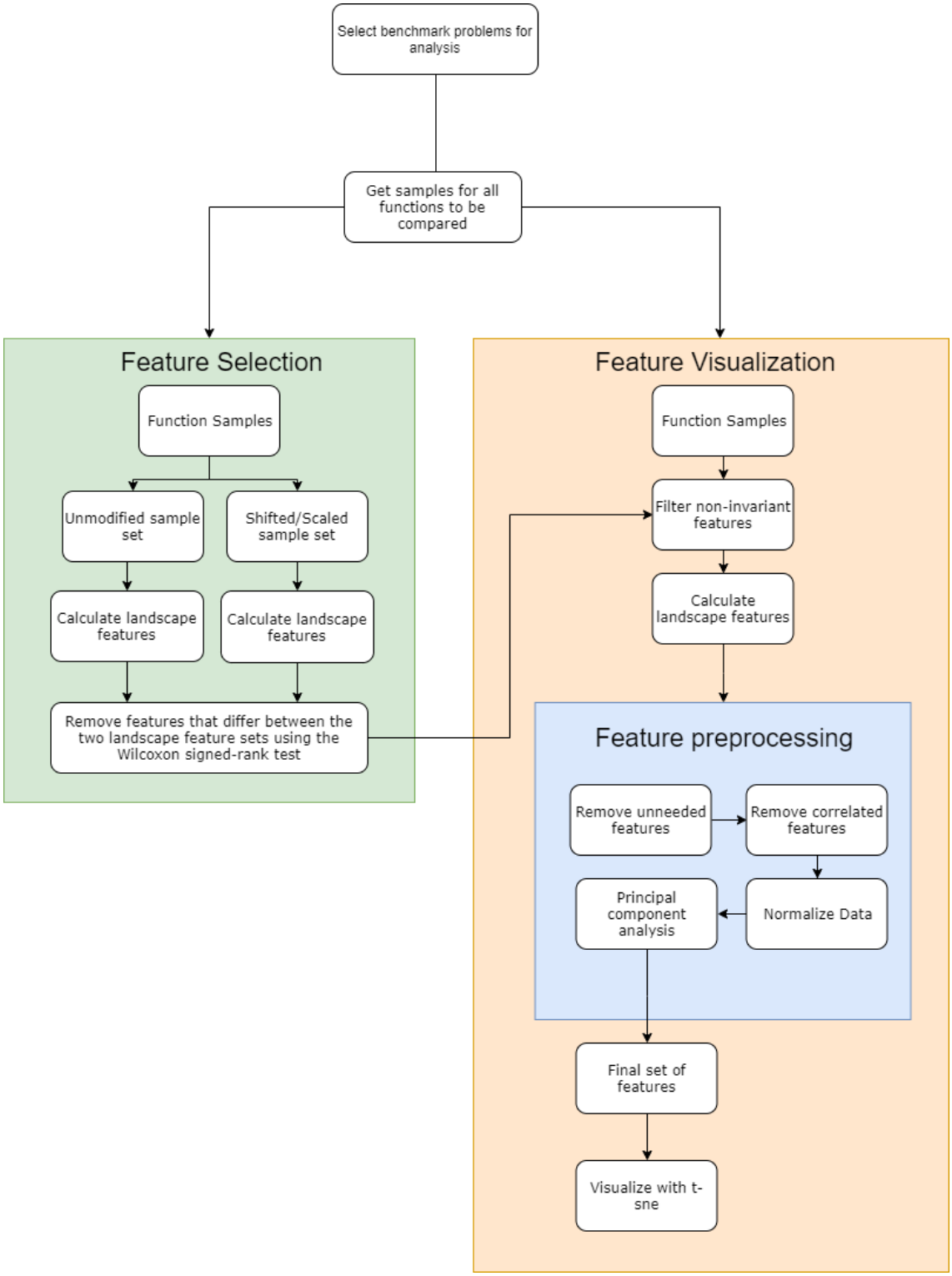

Figure 4.19: A diagram representing the experiment setup. First, non-invariant features are determined using a Wilcoxon test. Then these features are excluded in the comparison and visualization.

4.2.2.1 Feature Selection and Preprocessing

After calculating the landscape features, we first removed all features that produced only constant values or provided only irrelevant information such as the time it took the library to calculate features. After this process, we were left with 49 features out of the original 101. These 49 features are presented in Table 4.5.

Table 4.5: Landscape features left after removing constant and unnecessary features,

Feature Names 1–25	Feature Names 26–49
ela_distr.kurtosis	cm_angle.dist_ctr2best.mean
ela_distr.number_of_peaks	cm_angle.dist_ctr2worst.mean
ela_meta.lin_simple.adj_r2	cm_angle.angle.mean
ela_meta.lin_simple.intercept	cm_grad.mean
ela_meta.lin_simple.coef.min	ela_distr.skewness
ela_meta.lin_simple.coef.max	disp.diff_mean_25
ela_meta.lin_simple.coef.max_by_min	disp.diff_median_02
ela_meta.lin_w_interact.adj_r2	disp.diff_median_05
ela_meta.quad_simple.adj_r2	disp.diff_median_10
ela_meta.quad_simple.cond	disp.diff_median_25
ela_meta.quad_w_interact.adj_r2	limo.avg_length.reg
ic.h.max	limo.length.mean
ic.eps.max	limo.ratio.mean
ic.m0	nbc.nn_nb.sd_ratio
disp.ratio_mean_02	nbc.nn_nb.mean_ratio
disp.ratio_mean_05	nbc.nn_nb.cor
disp.ratio_mean_10	nbc.dist_ratio.coeff_var
disp.ratio_mean_25	nbc.nb_fitness.cor
disp.ratio_median_02	pca.expl_var.cov_init
disp.ratio_median_05	pca.expl_var.cor_init
disp.ratio_median_10	pca.expl_var_PC1.cov_x
disp.ratio_median_25	pca.expl_var_PC1.cor_x
disp.diff_mean_02	pca.expl_var_PC1.cov_init
disp.diff_mean_05	pca.expl_var_PC1.cor_init
disp.diff_mean_10	

The first goal of the feature selection stage was to determine which features are not invariant to scaling or shifting. Our analysis revealed that 26 out of 49 features were not invariant to these transformations. This leaves us with a total of 23 invariant features. These invariant features are listed in Table 4.6. Figure 4.20 shows a visual representation of p-values obtained when comparing landscape feature values between original and transformed problems on the 2014 dataset. We can see that for some features, p-values are very low, much lower than the standard threshold of 0.05, while for some, the p-value is much higher than the threshold (about 0.2). Only a small number of features is borderline in the sense that their p-value is close to the significance level of 0.05.

As a final step, we checked which of these features are highly correlated with one another. This was done by using two approaches. First by using Pearson correlation, and the second by using PCA.

The correlation analysis using Pearson correlation was carried out by computing the Pearson correlation matrix, selecting pairs of features with absolute correlation of more than 0.95 , and for each pair removing the feature with the higher mean correlation with

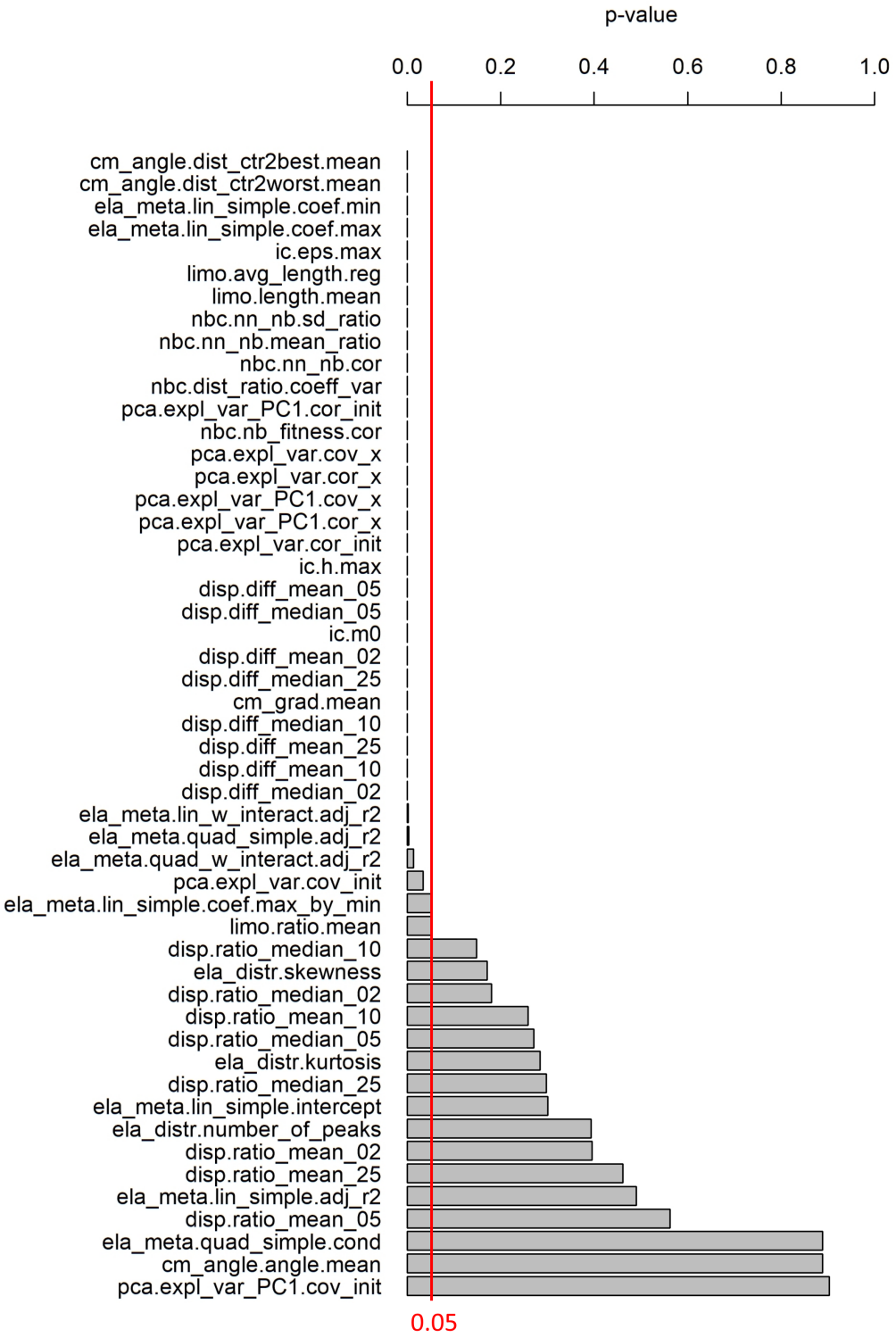

Figure 4.20: p-values of individual landscape feature comparisons showing which features are invariant to scaling and shifting produced by a Wilcoxon test. A p-value lower than 0.05 means that the feature is invariant to these two transformations. The red line indicates the value of 0.05.

Table 4.6: Landscape features invariant to shift and scale. The bolded features are those that remained after eliminating highly correlated features.

Feature Name
cm_angle.angle.mean
ela_distr.skewness
ela_distr.kurtosis
ela_distr.number_of_peaks
ela_meta.lin_simple.adj_r2
ela_meta.lin_simple.intercept
ela_meta.lin_simple.coef.min
ela_meta.quad_w_interact.adj_r2
ela_meta.quad_simple.adj_r2
ela_meta.lin_w_interact.adj_r2
disp.ratio_mean_02
disp.ratio_median_25
nbc.nb_fitness.cor
pca.expl_var_PC1.cov_init
pca.expl_var.cov_init
disp.ratio_mean_05
disp.ratio_mean_10
disp.ratio_mean_25
disp.ratio_median_02
disp.ratio_median_05
disp.ratio_median_10
pca.expl_var_PC1.cor_init

all other features. The relatively high value for the correlation cutoff was chosen because a large number of features displayed very high correlation. In total, this removed seven out of 23 remaining features, leaving us with a total of 16 bold features listed in Table 4.6.

Figure 4.21 shows the results of the Pearson correlation. We can see that the vast majority of the removed features belong to the *disp.ratio* group, which are all correlated with each other. Despite their large correlation, our procedure retained two out of the eight features in the group, instead of just one, as the correlation between some of these features was just below the chosen threshold. We can also see that the features *pca.expl_var.cor_init* and *pca.expl_var.cov_init* are strongly inversely correlated with the features *pca.expl_var_PC1.cor_init* and *pca.expl_var_PC1.cov_init*, respectively. Only the feature *pca.expl_var_PC1.cor_init* was removed, as the correlation between the other two features was again just barely below the threshold.

Finally, we performed our analysis using PCA. This further reduced the data to a total of about five principal components, depending on the exact dataset. Figure 4.22 shows a visual representation of the first 19 principal components obtained when comparing the landscape features calculated on the combined set of CEC 2014 and GECCO problems. We can see that the first component explains about 25% of the variance, and the first three explain more than 50%. After that, the amount of explained variance decreases slowly with each additional component.

Before performing the complementarity experiments, the perplexity parameter of the t-sne visualization had to be determined. This parameter has a large effect on the produced visualization and is typically set to a value between 5 and 50. The best value is dependent

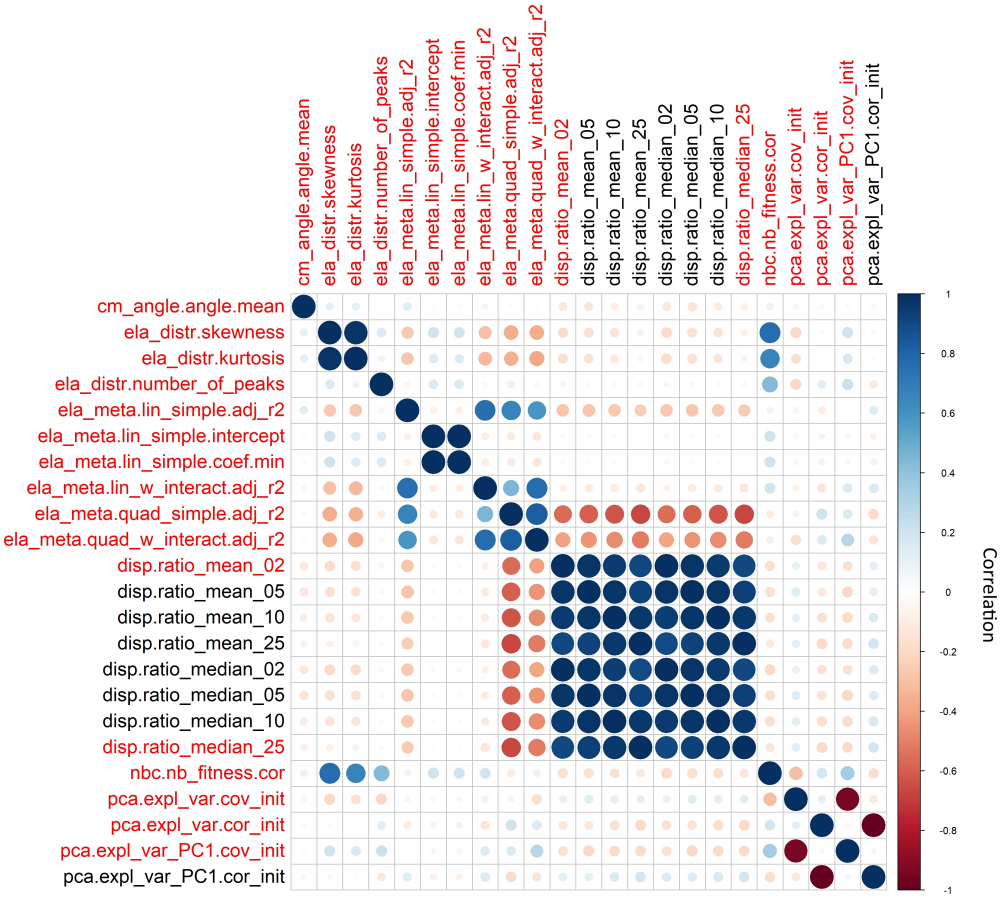

Figure 4.21: The results of the Pearson correlation calculation. Bigger and bolder circles represent higher correlation, which is either positive (blue) or negative (red). Red labels represent the features that were retained, while black labels represent features that were removed from the feature set.

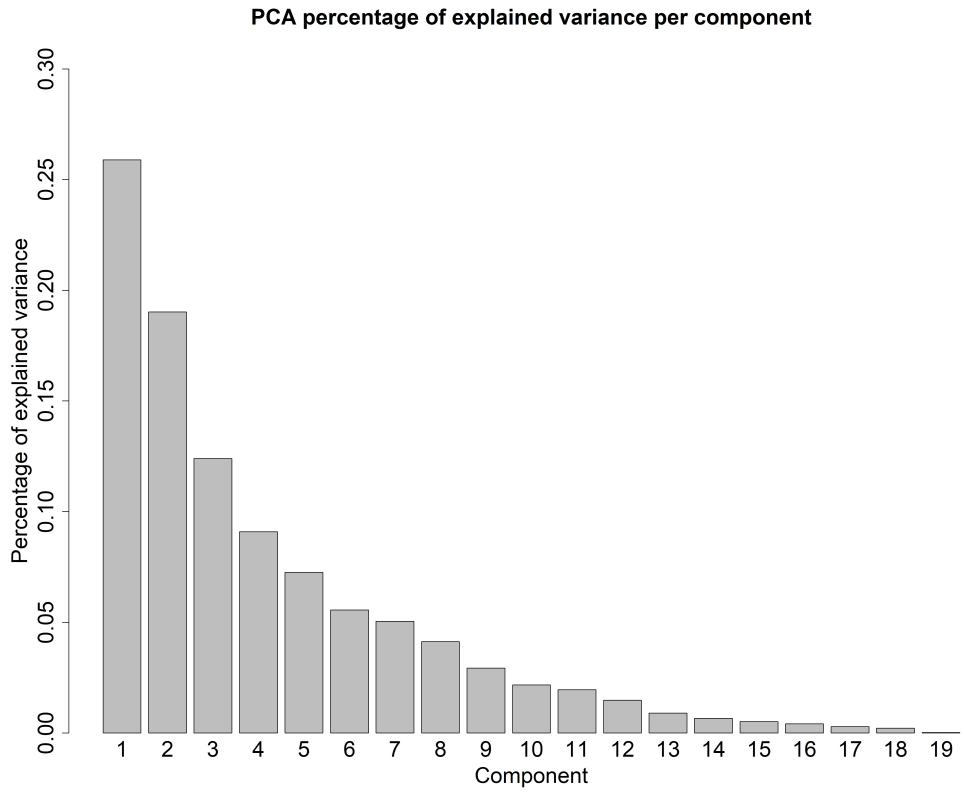

Figure 4.22: The amount of explained variance per component when performing PCA on the landscape features calculated on the combined set of 2014 CEC and GECCO problems.

on the data being visualized and cannot be determined automatically.

We attempted to tune this parameter automatically by running the experiment described in Section 4.2.2.2 with different perplexity values and measuring the root mean square error between the pairs of problems listed in Table 4.10. We performed 200 separate visualizations for perplexity values between 1 and 10. However, a Wilcoxon test showed no statistically significant differences between these perplexity values. However, larger perplexity values such as 20 showed statistically worse results. Because of this, we chose a perplexity parameter in the range between 1 and 10, specifically the value 5. This parameter value is used in all further experiments.

Figures 4.23 and 4.24 show example visualizations of the same data using the perplexity parameters of 5 and 20. We can see that the value of 20 splits data almost evenly across the visualization, which makes it impossible to determine any clusters of data.

To verify that excluded features produce improvements in the visualization, we performed the following experiment:

1. Pick a year of the CEC competitions to use for comparison. Create two sets of problem samples: one without any transformations applied and one with a random shift and scale transformation applied.

2. Calculate the invariance of the features as described in Section 4.1.1.

3. Use the invariant features from Step 2 to visualize the data in Step 1.

Figure 4.25 shows the result of the scaling and shifting operation described in Step 1. The red dots show the original problem samples, while the blue dots show the scaled and shifted samples. The comparison uses the CEC 2014 Rotated High Conditioned Elliptic function. We can see that the overall shape of the problem remains similar, but the distance between samples and the location of the optimum differ.

Here, we note that the shifting and scaling operations we will use in our invariance experiments transform problem samples directly. This makes the transformations that we will consider in this chapter different from transformations that are used by for example the BBOB benchmarks to construct problem instances. This is because the BBOB benchmarks instead perform the shifting and scaling transformations on the underlying problem, rather than on the problem samples.

For this experiment, we selected the problems from the 2014 CEC competition to use for the comparison. The results of these experiments done using the sample size of $n_{samp} = 50D$ are shown in Figures 4.26, 4.27, and 4.28. In these figures, the red numbers represent the original problems, while the black numbers represent the corresponding transformed problems. Ideally, the same numbered problems of different colors should be visualized close together.

Figure 4.26 shows the t-sne visualization after calculating the base landscape features but before any preprocessing. Figure 4.27 shows what happens if we perform PCA on the full feature set, without first excluding non-invariant features. As we can see, the visualization almost cleanly splits the dataset into two groups: the original problems and the problems that had the scale and shift transformation applied to them.

Finally, Figure 4.28 shows the visualization without the non-invariant landscape features after performing normalization and PCA. Here, we can see that the same numbered problems stay relatively close to one another. While the "equal" problems are not exactly on top of one another, the resulting visualization is much better than the visualization that included the non-invariant features in Figure 4.27.

However, problem 19 does not appear closest to one another in this visualization. To further investigate why some problems still appear apart on the visualization, we exam-

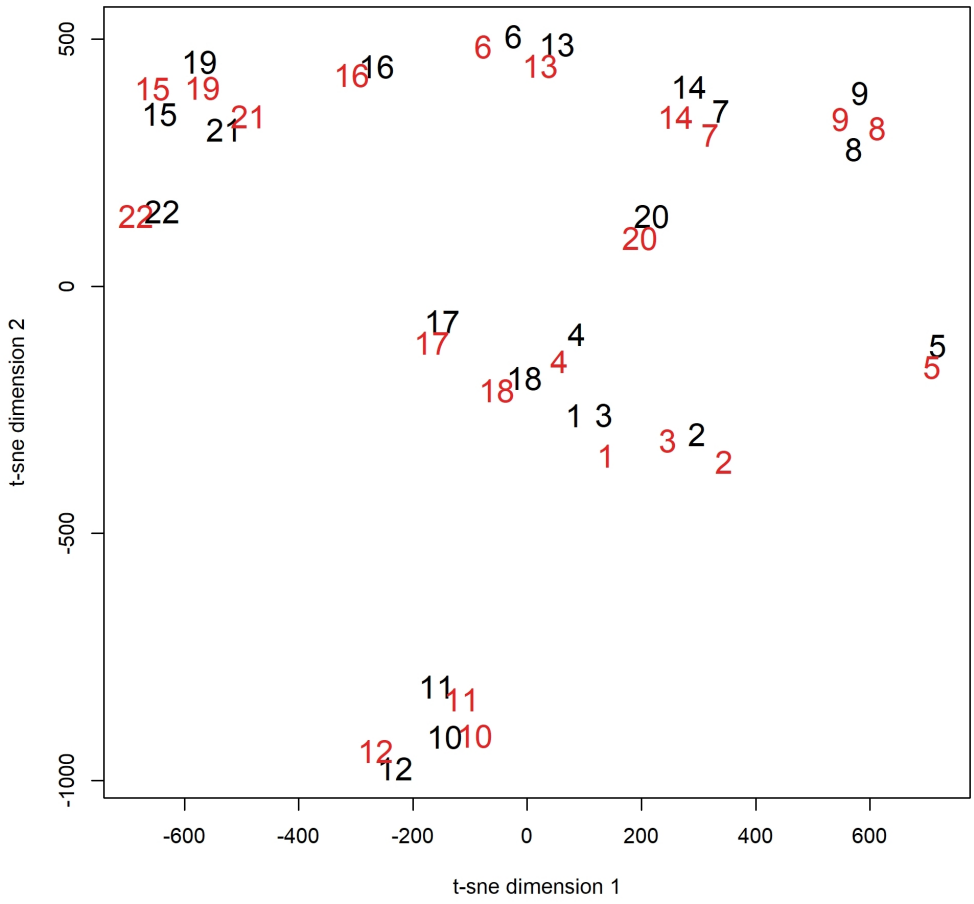

Figure 4.23: The t-sne visualization of two different set of landscape features using a perplexity parameter of 5. The black numbers represent the features calculated on the original samples of CEC 2014 problems, and the red numbers represent the features calculated on the shifted and scaled samples.

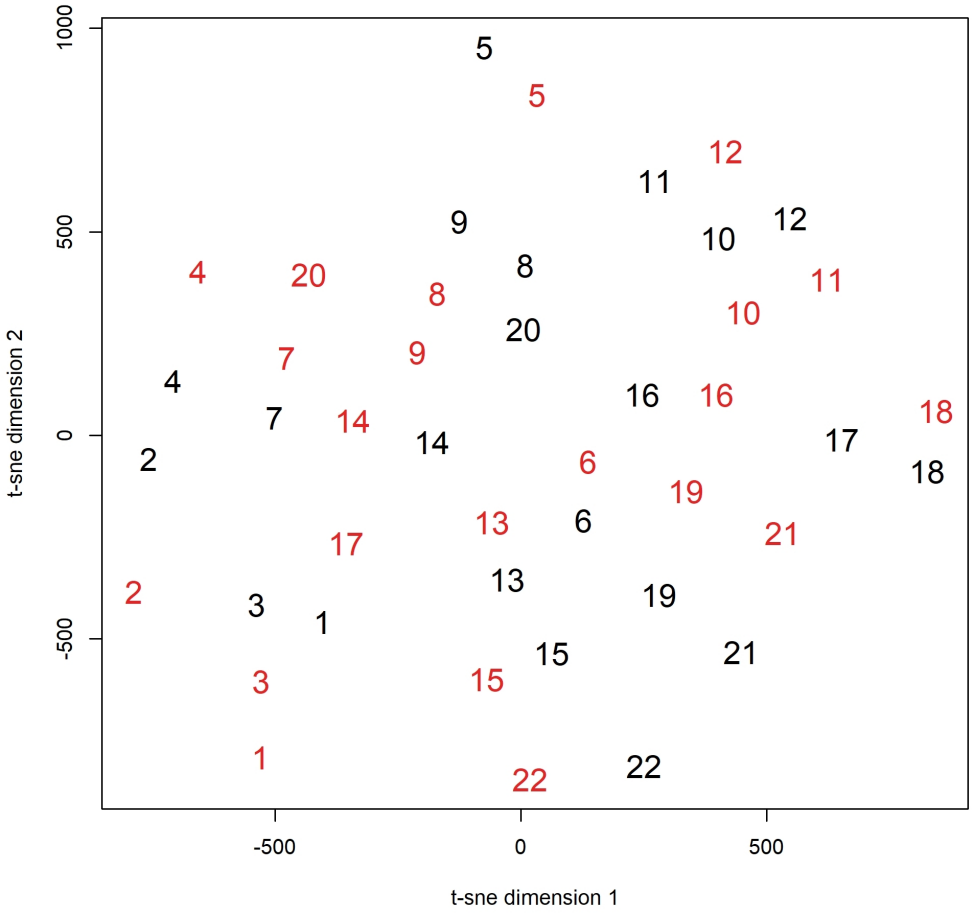

Figure 4.24: The t-sne visualization of two different set of landscape features using a perplexity parameter of 20. This produces a worse visualization, with problems evenly distributed and without any structure. The black numbers represent the features calculated on the original samples of CEC 2014 problems, and the red numbers represent the features calculated on the shifted and scaled samples.

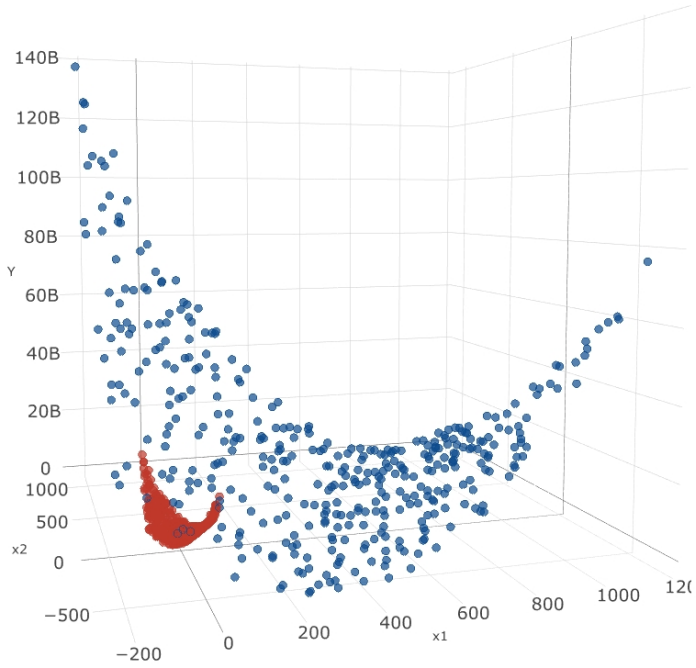

Figure 4.25: A scatterplot showing the effect of the transformations used to determine non-invariant features. Red points represent the original problem, while blue points represent the transformed problem.

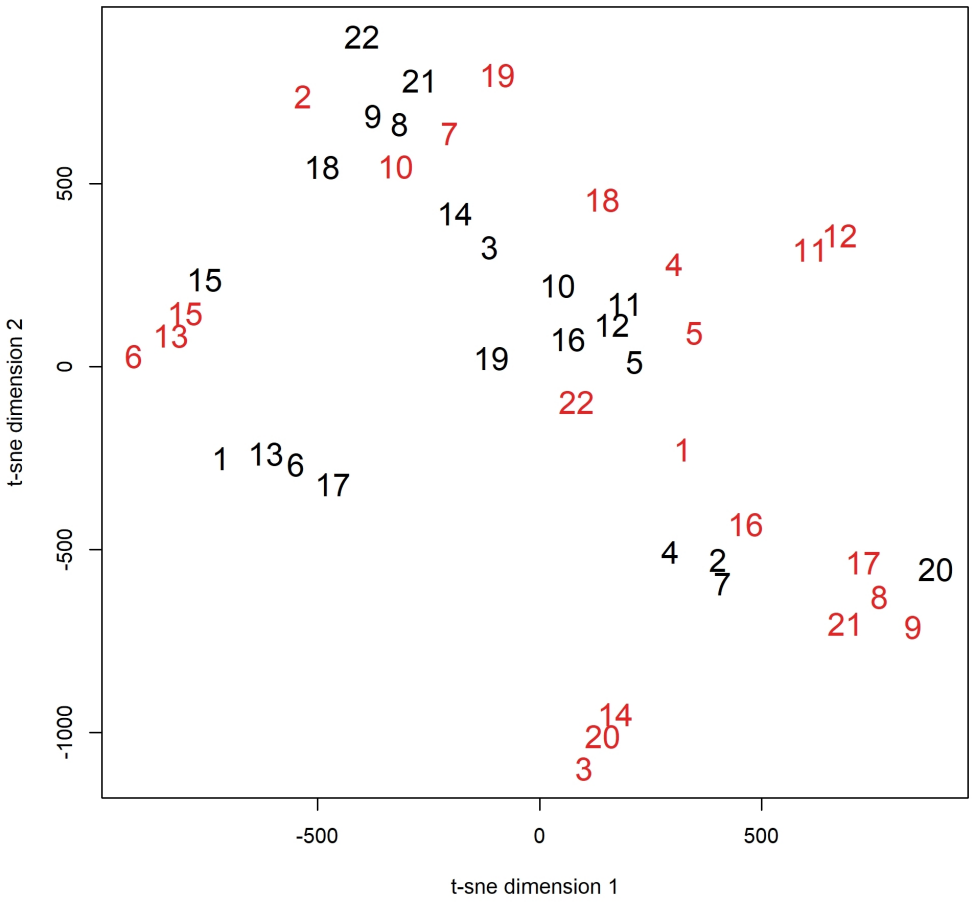

Figure 4.26: A visualization of two sets of problems from the CEC 2014 problem set, with the original problems in red, and the shifted and scaled problems in black. The visualization is based on raw ELA data, with no processing applied.

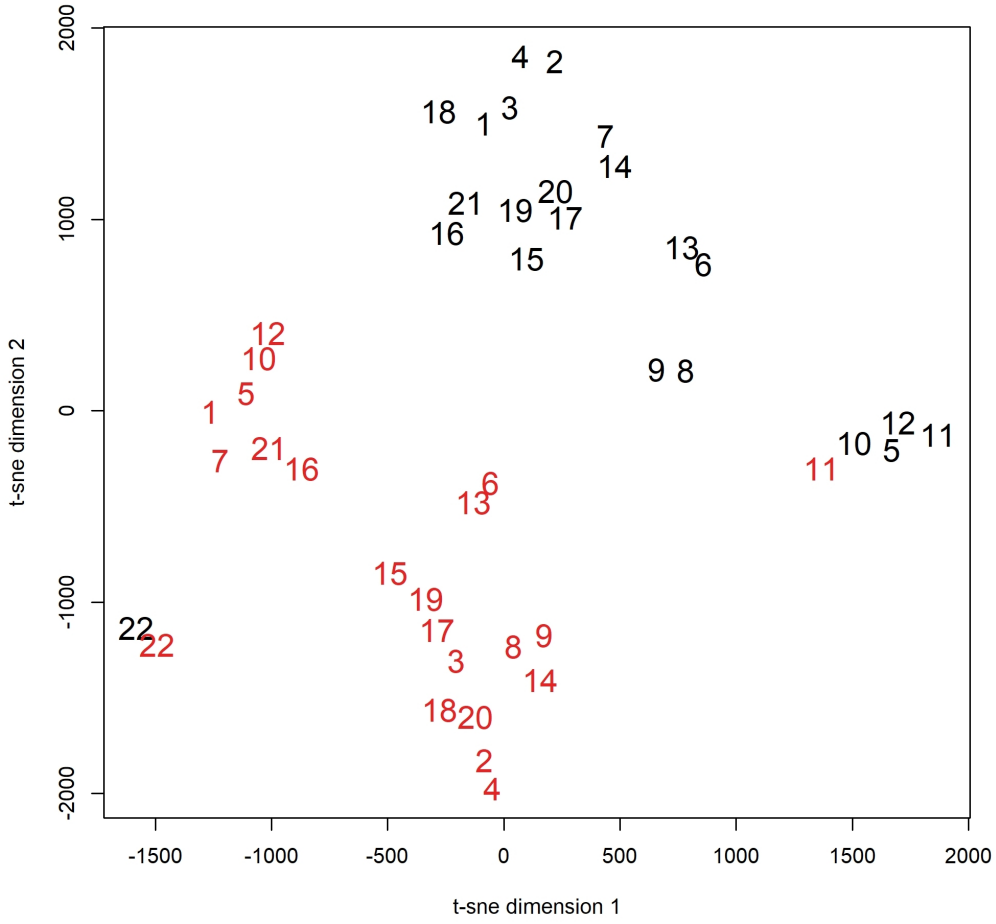

Figure 4.27: A visualization of two sets of problems from the CEC 2014 problem set, with the original problems in red, and the shifted and scaled problems in black. The visualization is based on ELA data, with PCA applied.

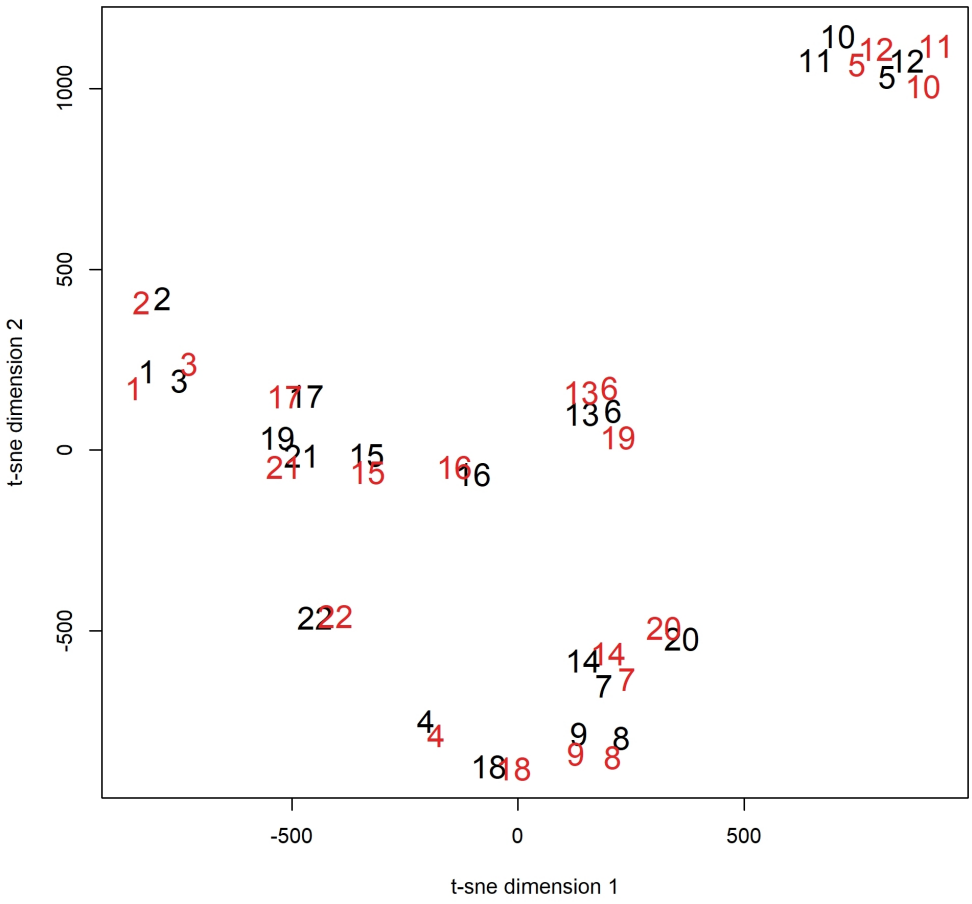

Figure 4.28: A visualization of two sets of problems from the CEC 2014 problem set, with the original problems in red, and the shifted and scaled problems in black. The visualization is based on raw ELA data, with PCA applied and the features that are not invariant to shifting and scaling removed.

ined also the influence of the sample size parameter. We performed the same experiment described above on two-dimensional data on several additional sample sizes from $n_{samp} = 100D$ to $n_{samp} = 1000D$. We determined that the sample size of $n_{samp} = 200D$ shown in Figure 4.29 provided the best results, as it also visualizes the problem 19 close together. For this reason, we have decided to use the sample size of $n_{samp} = 200D$ for all further experiments.

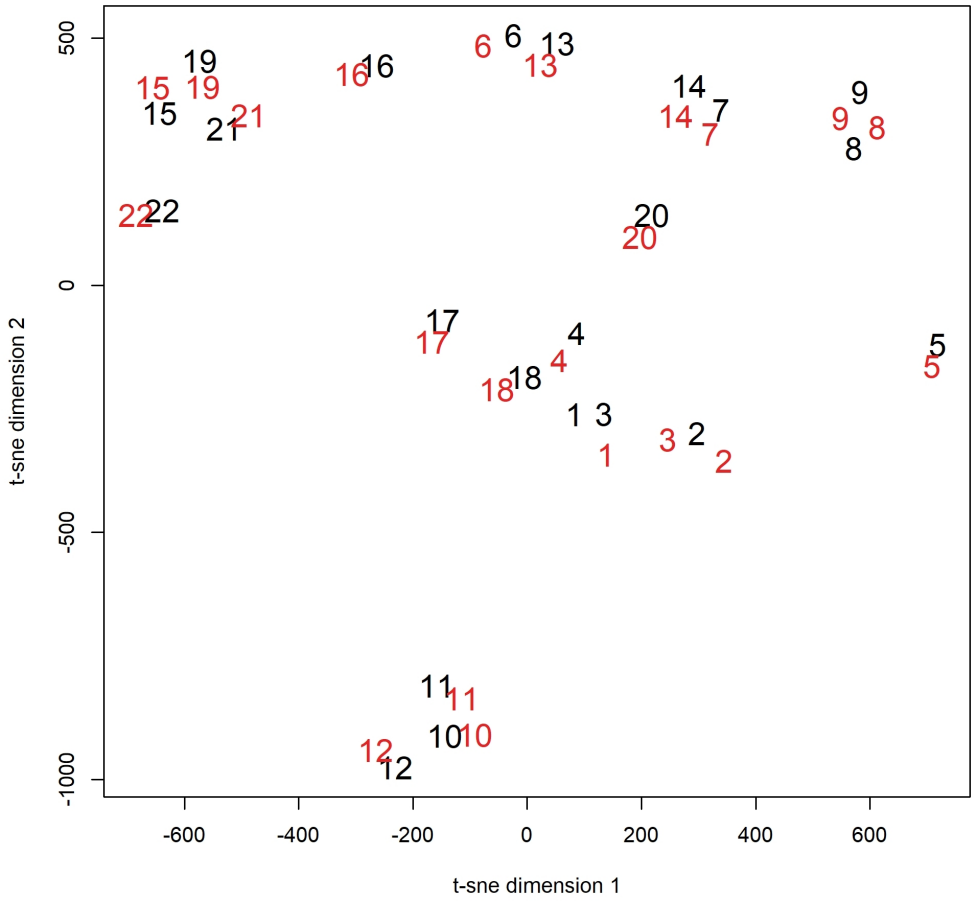

Figure 4.29: A visualization of two sets of problems from the CEC 2014 problem set, with the original problems in red, and the shifted and scaled problems in black. The visualization is based on raw ELA data, with PCA applied and the features that are not invariant to shifting and scaling removed, and using a sample size of $n_{samp} = 200D$ rather than $n_{samp} = 50D$, using two-dimensional data.

We also performed the same experiment on 10-dimensional data and achieved similar results. Figure 4.30 shows the visualization of 10-dimensional data at a sample size of $200D$. However, even the best visualization still cannot visualize some problems close together, in particular the problem number 20. We believe that this could be happening due to the Wilcoxon process failing to exclude certain features that are not invariant to scaling or transformation only for a small number of problems.

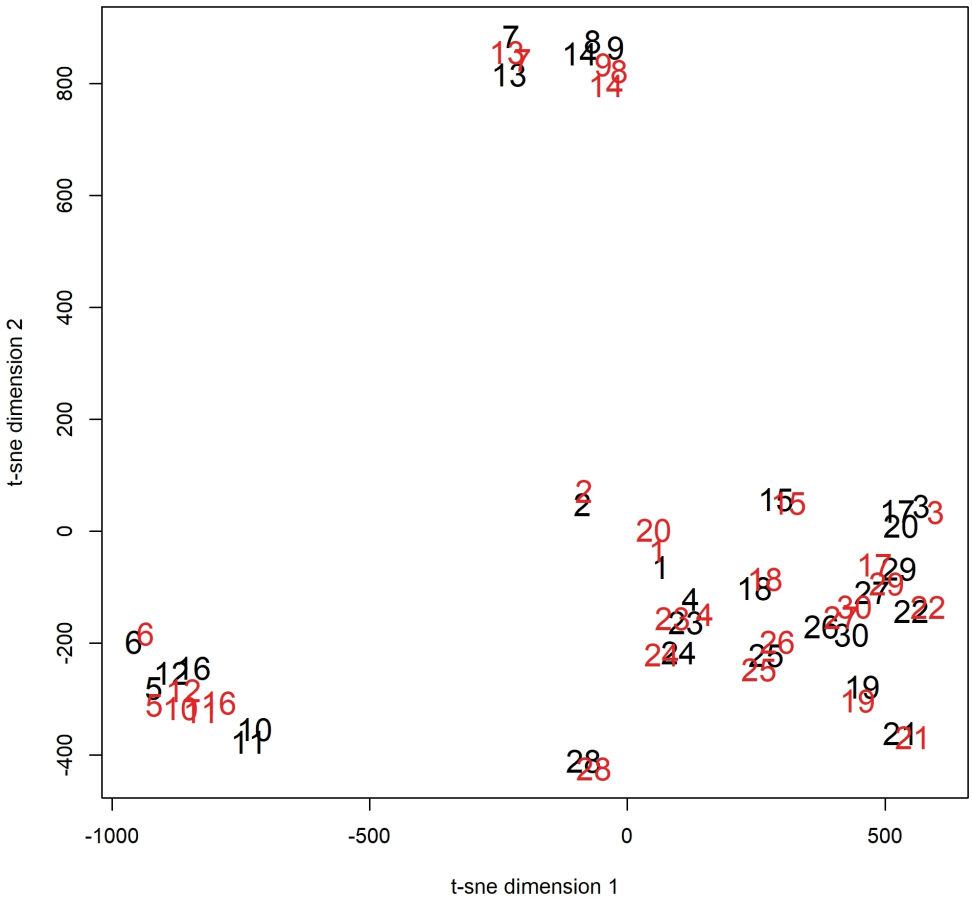

Figure 4.30: A visualization of two sets of problems from the CEC 2014 problem set, with the original problems in red, and the shifted and scaled problems in black. The visualization is based on raw ELA data, with PCA applied and the features that are not invariant to shifting and scaling removed, and using a sample size of $n_{samp} = 200D$ rather than $n_{samp} = 50D$, using 10-dimensional data.

4.2.2.2 CEC 2014 Versus CEC 2015

The preliminary experiments described in the preceding subsection show that our methodology is capable of visualizing the same problems close together even under the transformations of shifting and scaling when comparing two sets of identical problems.

In this subsection, we describe a slightly more advanced version of the experiment: comparing two sets of problems from different years of the CEC benchmark. In particular, we focus on the results of the comparison between the CEC years of 2014 and 2015. This experiment differs from the previous one in two key ways:

1. Since different years of the CEC competitions use different problems, the set of problems is no longer completely identical. However, there are still some problems that are present in both years and should be visualized close together.

2. The problems feature additional transformations that were not taken into account in the previous subsection. These include for example the scrambling and rotation transformations. A scrambling rotation changes the order of the original values $[x_1, x_2, ..., x_n]$ into $[x_{p1}, x_{p2}, ..., x_{pn}]$, where $p1, ..., pn$ is a permuted sequence of numbers $1, ..., n$. A rotation transformation, on the other hand, can change some properties of the underlying problem, such as its separability. In addition, these transformations are applied to the underlying problem, while the transformations in the previous subsection were applied to the samples after the results are calculated.

The results of this experiment are shown in Figure 4.31 for two-dimensional data and Figure 4.32 for 10-dimensional data. The list of problem numbers and their corresponding problems are available in Tables 4.7 and 4.8 for the 2014 problem set, and Table 4.9 for the 2015 problem set. For convenience, Table 4.10 shows which problems from the year 2014 are the same as problems from the year 2015. Note that some of the problems from both benchmarks are not defined for two dimensions, so the two-dimensional comparison will have fewer problems.

We can see that the experiment is able to place almost all identical problems close together.

In the two-dimensional visualization, we can see that similar problems are visualized close to one another. In particular the problem pairs (red, black): (3,5), (4,9), and (5,11). Also of interest are problems one and two of both problem sets. These are the High Conditioned Elliptic function and the Bent Cigar function in both of the problem sets. However, in two dimensions, both of these problems are actually identical. So we would expect to see all four of these problems visualized close together. From the visualization, we can see that three out of four of them are visualized right next to one another, while the fourth problem, the black 1, is not. To explain this, Figures 4.33 and 4.34 show two different scatterplot comparisons. Figure 4.33 shows the comparison between the red problem 1 and black problem 2, which are visualized close to one another. Figure 4.34 shows the comparison between red 1 and black 1, which are consequently correctly visualized further apart.

In the 10-dimensional data shown in Figure 4.32, we can see that the pairs (red, black): (3,5), (5,11), and (2,2) are still visualized close together. However, the pair (4,9) is no longer very close but still appears relatively close. Unfortunately, we are not able to visualize 10-dimensional data. However, we believe that a similar situation to the one described above might be occurring here: that a transform, particularly rotation, is causing the two problems to be visualized far apart. On the other hand, we can see that problems one and two are now correctly visualized close to one another. In 10 dimensions, these two

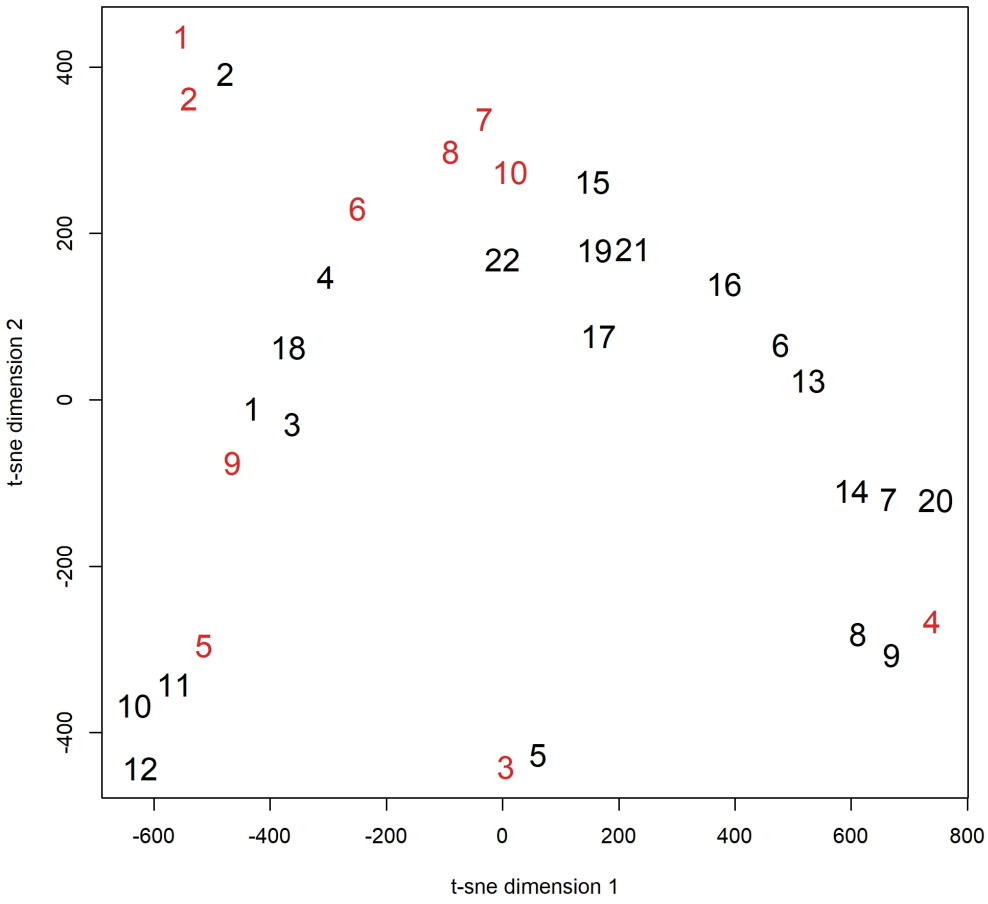

Figure 4.31: t-sne visualization of CEC 2014 (black) vs CEC 2015 (red) problems, two-dimensional data.

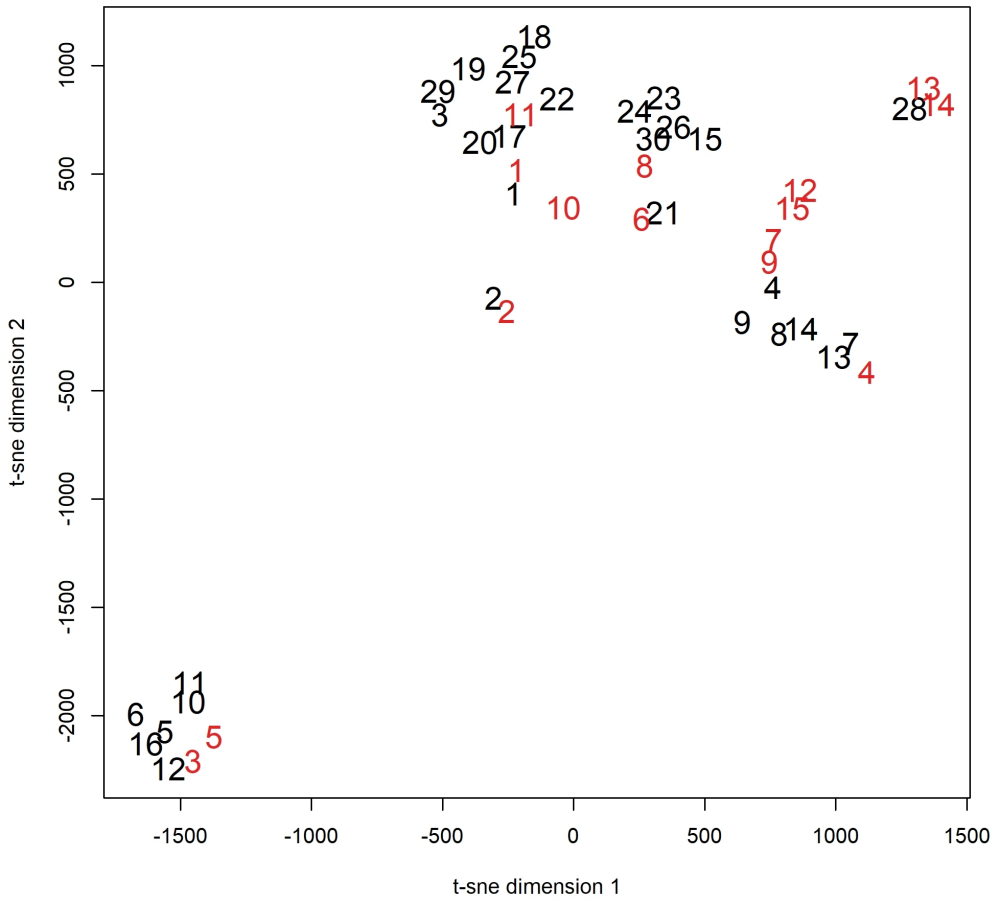

Figure 4.32: t-sne visualization of CEC 2014 (black) vs CEC 2015 (red) problems, 10-dimensional data.

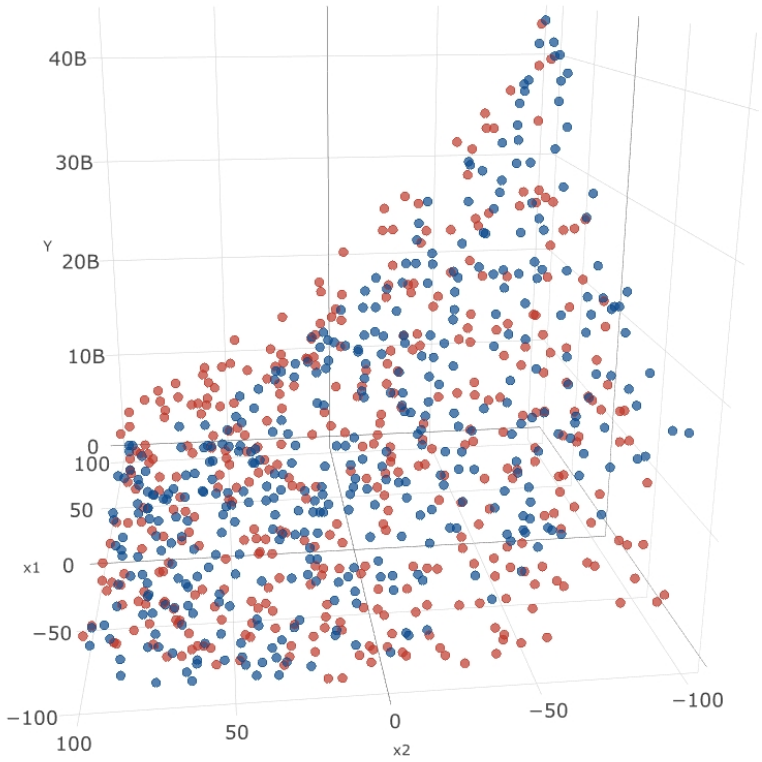

Figure 4.33: A scatterplot comparison of the 2014 CEC problem 2 (red) and 2015 problem 1 (blue).

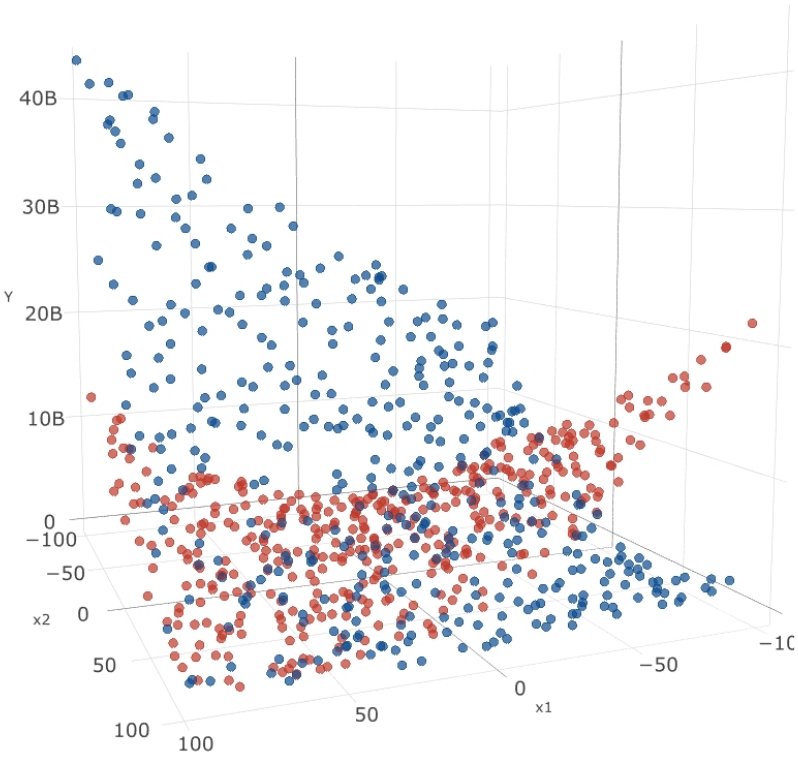

Figure 4.34: A scatterplot comparison of the 2014 CEC problem 1 (red) and 2015 problem 1 (blue).

Table 4.7: CEC 2014 problems 1-15.

Problem id $D = 10$	Problem id $D = 2$	Problem name
1	1	Rotated High Conditioned Elliptic Function
2	2	Rotated Bent Cigar Function
3	3	Rotated Discus Function
4	4	Shifted and Rotated Rosenbrock's Function
5	5	Shifted and Rotated Ackley's Function
6	6	Shifted and Rotated Weierstrass Function
7	7	Shifted and Rotated Griewank's Function
8	8	Shifted Rastrigin's Function
9	9	Shifted and Rotated Rastrigin's Function
10	10	Shifted Schwefel's Function
11	11	Shifted and Rotated Schwefel's Function
12	12	Shifted and Rotated Katsuura Function
13	13	Shifted and Rotated HappyCat Function
14	14	Shifted and Rotated HGBat Function
15	15	Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function

problems are no longer identical, and this is visible in the visualization, as the problems are no longer placed directly next to one another.

4.2.2.3 CEC Versus BBOB

For the final comparison, we wanted to compare the problems of two different benchmarks: the CEC competitions and the BBOB workshops. This presents the following additional challenges compared to the previous subsection.

1. Since we are comparing two completely different benchmarks, there can be differences in the way they design their benchmark problems. Even problems that share the same name between the two benchmarks (for example, a Schwefel's function) might differ, for example, due to different parameter selection.

2. The two benchmarks might apply completely different transformations to their benchmark problems. For example, the CEC competitions apply a shuffle transformation, while the BBOB workshops do not.

3. The sampling space of each problem might be different. Sampling the same problem at two different ranges might produce completely different results even though we are sampling the same underlying problem.

For this experiment, we present the results in two different ways. In the first part, we compare only the problem set from the 2014 CEC competition with the problem set of the BBOB workshops. We limit ourselves to only a part of the CEC problems for visibility reasons so that both the CEC and the BBOB problem sets will contain a similar amount of problems. In the second part, we compare the BBOB workshops problem set with the entire CEC problem set from the years 2014-2017, which contains a total of 102 problems.

Table 4.8: CEC 2014 problems 16-30.

Problem id $D = 10$	Problem id $D = 2$	Problem name
16	16	Shifted and Rotated Expanded Schaffer's F6 Function
17	-	Hybrid Function 1 (N=3)
18	-	Hybrid Function 2 (N=3)
19	-	Hybrid Function 3 (N=4)
20	-	Hybrid Function 4 (N=4)
21	-	Hybrid Function 5 (N=5)
22	-	Hybrid Function 6 (N=5)
23	17	Composition Function 1 (N=5)
24	18	Composition Function 2 (N=3)
25	19	Composition Function 3 (N=3)
26	20	Composition Function 4 (N=5)
27	21	Composition Function 5 (N=5)
28	22	Composition Function 6 (N=5)
29	-	Composition Function 7 (N=3)
30	-	Composition Function 8 (N=3)

The results of the first part of the experiment are shown in Figure 4.35 for two-dimensional data and Figure 4.36 for 10-dimensional data. The list of problem numbers and their corresponding problems are available in Table 4.12 for the BBOB problem set, and Tables 4.7, 4.8 for the CEC problem set. For convenience, Table 4.11 shows which problems from the CEC benchmarks are identical in name to which problems from the BBOB benchmark.

From the figures, very few problems that share the same names are visualized close together. Only the discus function (black 3, red 11) appears somewhat close, while others are far apart.

As before, we use scatterplots to investigate these differences. Note that in these figures, the Y (result) values have been scaled to fit into the range of [0,1]. This was done to make the comparison easier, as the two benchmark sets use very different scales for the Y-axis. In order to eliminate this different Y scale being the cause of the poor performance of the experiments, we also reran the experiments using scaled Y values for both benchmark sets. However, there was no difference in the obtained results.

Figure 4.37 shows the scatterplot of the Shifted Schwefel's Function, which is the problem number 10 for CEC (black) and the problem number 20 for BBOB (red) problems. As we can see, problems that share the same name across both benchmarks nonetheless appear very different in scatterplots.

Figure 4.38 compares the CEC (black) problem number 19 with the BBOB (red) problem 13. Here, the problems have similar scatterplots, which explains why they are visualized close together. We observed similar effects in the other problems. More examples are available in our journal article related to this chapter (Škvorc et al., 2020).

This indicates that there is some fundamental design difference between these problems, either in various parameters or in the overall benchmark design. One such example is the search area. The CEC benchmarks define the area of interest to the range of $[-100, 100]$, while the BBOB workshops define it to the area of $[-5, 5]$, meaning that the global optimum is guaranteed to be located in this area, and that optimization algorithms should focus on it. To deal with this discrepancy, our experiments used the range of $[-100, 100]$ for both

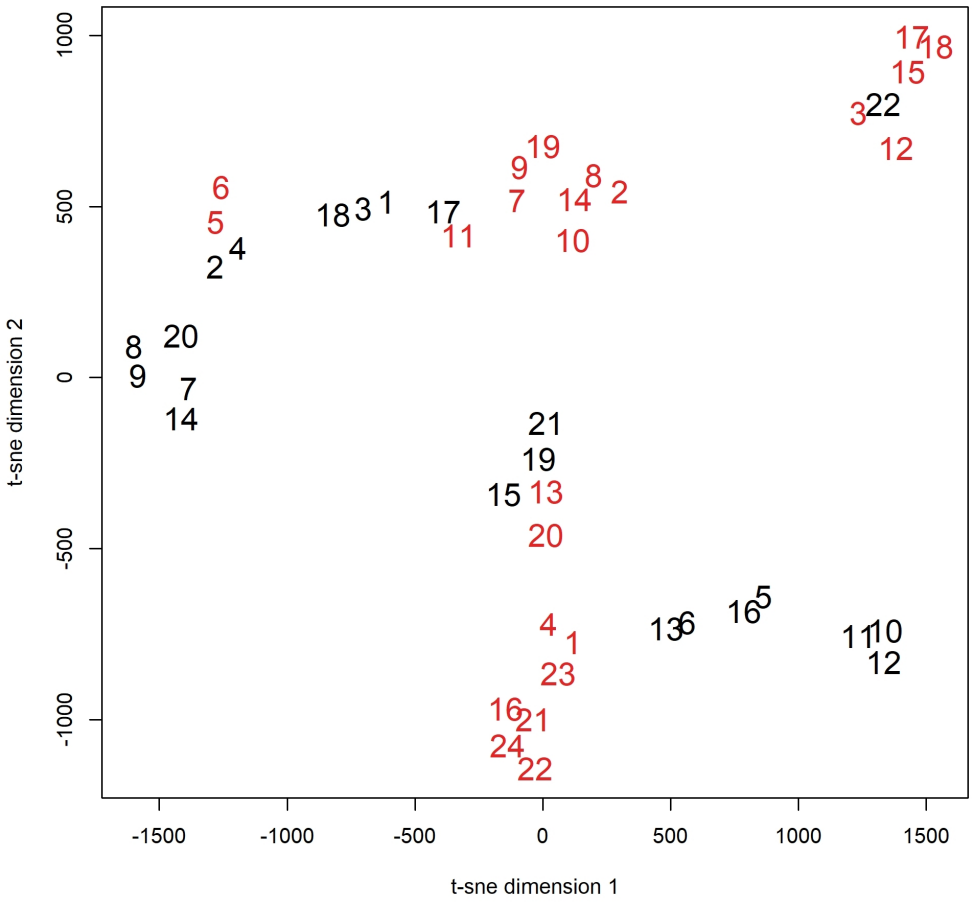

Figure 4.35: t-sne visualization of CEC vs BBOB two-dimensional problems, using only 2014 CEC problems.

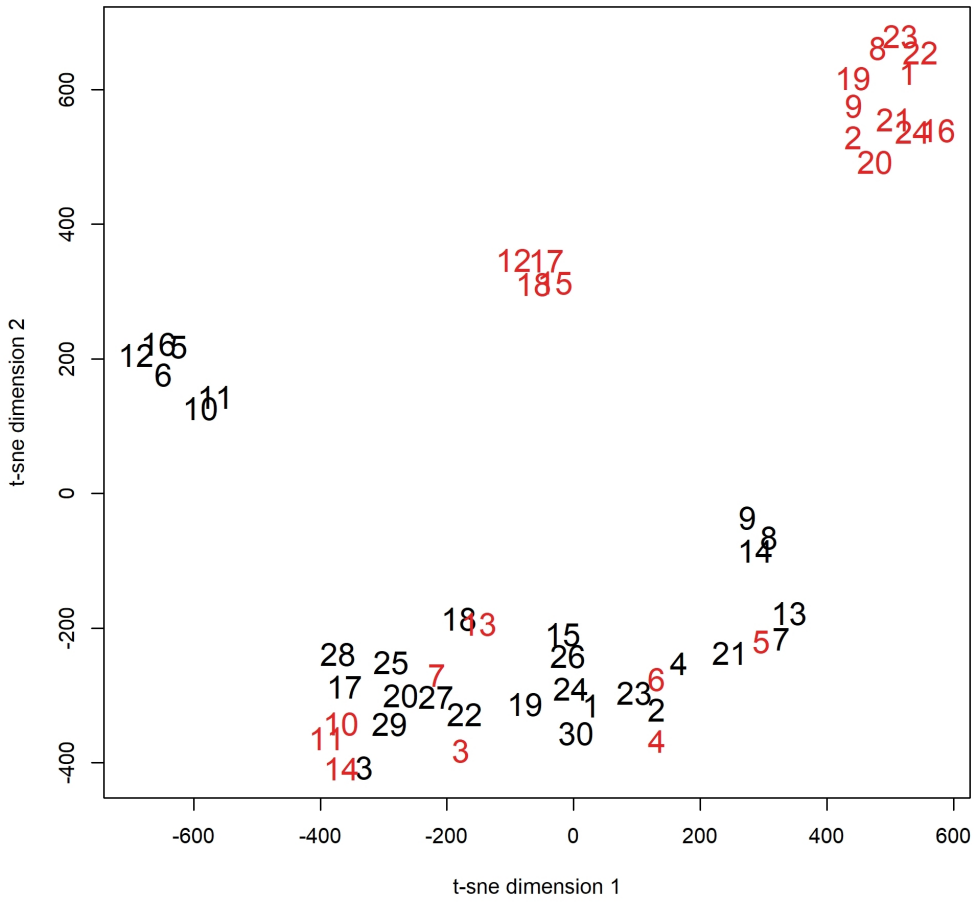

Figure 4.36: t-sne visualization of CEC vs BBOB 10-dimensional problems, using only 2014 CEC problems.

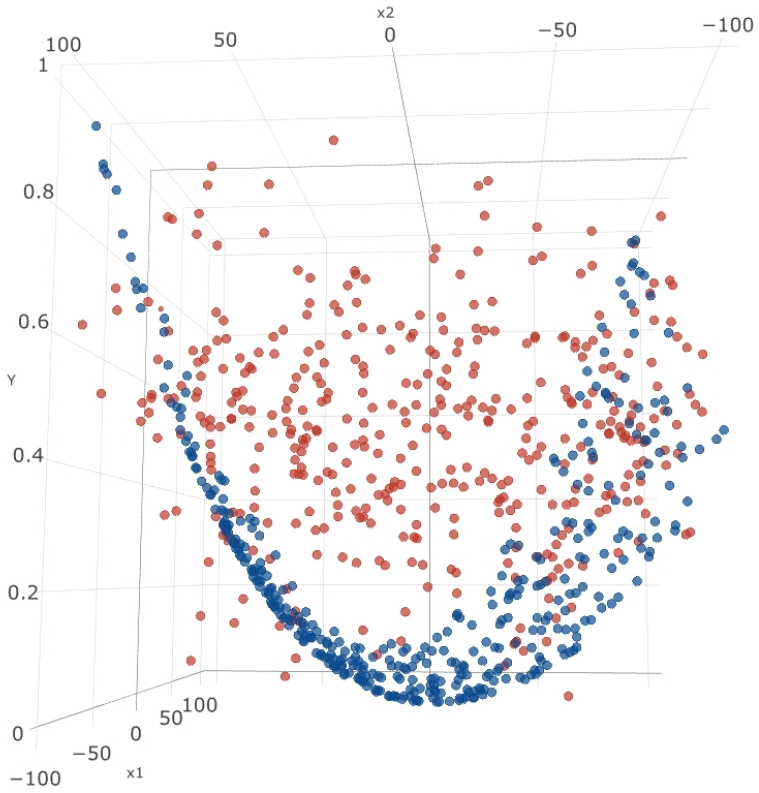

Figure 4.37: CEC problem 10 (red) vs BBOB problem 20 (blue), which are visualized far apart.

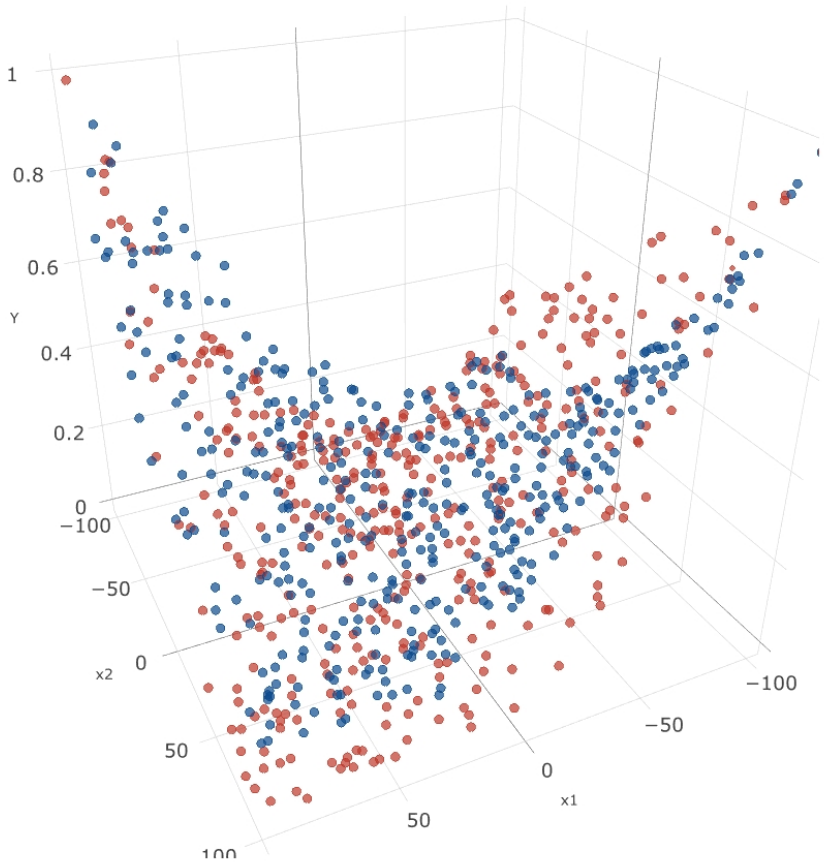

Figure 4.38: CEC problem 19 (red) vs BBOB problem 13 (blue), which are visualized close together.

Table 4.9: CEC 2015 problems.

Problem id $D = 10$	Problem id $D = 2$	Problem name
1	1	Rotated High Conditioned Elliptic Function
2	2	Rotated Cigar Function
3	3	Shifted and Rotated Ackley's Function
4	4	Shifted and Rotated Rastrigin's Function
5	5	Shifted and Rotated Schwefel's Function
6	-	Hybrid Function 1 (N=3)
7	-	Hybrid Function 2 (N=4)
8	-	Hybrid Function 3 (N=5)
9	6	Composition Function 1 (N=3)
10	-	Composition Function 2 (N=3)
11	7	Composition Function 3 (N=5)
12	8	Composition Function 4 (N=5)
13	-	Composition Function 5 (N=5)
14	9	Composition Function 6 (N=7)
15	10	Composition Function 7 (N=10)

Table 4.10: Common problems between the CEC 2014 and 2015 problem sets.

Problem id 2015	Problem id 2014	Problem name
1	1	Rotated High Conditioned Elliptic Function
2	2	Rotated Cigar Function
3	5	Shifted and Rotated Ackley's Function
4	9	Shifted and Rotated Rastrigin's Function
5	11	Shifted and Rotated Schwefel's Function

the CEC and the BBOB problems. To examine if the choice of the range was an issue, we conducted a separate experiment where we used the range of $[-5, 5]$ for the BBOB problems and the range of $[-100, 100]$ for the CEC problems. However, this experiment performed similarly to the original one, which indicates that the choice of the search range did not affect the results.

Another difference might be the transformations these benchmarks apply to the problems, such as rotation, shifting, and scaling. While we have tried to eliminate the effect of shifting and scaling, this had to be done after the problem was already calculated, while the benchmarks perform shifting and scaling on the problems before calculating results.

Finally, Figures 4.39 and 4.40 show the complementarity analysis results for problems from all years of the CEC and BBOB problem sets. Due to the large number of problems, these figures are mainly used to demonstrate the overall distribution of the problems between the two benchmarks. We can see that the visualization places the two different benchmarks into somewhat separate groups. As we have discussed above, this seems to indicate some sort of design difference in the problems of the two benchmarks.

In order to account for different possible shift and scale transformations, we attempted to automatically scale and shift several of the problems used for the above scatterplot comparisons. We used Differential Evolution (K. Price et al., 2006) as implemented in the R library DEoptim (Ardia et al., 2016) with default parameters to optimize the factors for scaling and shifting transformations. The mean pairwise distance between the closest points was used as a fitness function. However, this did not provide a big improvement.

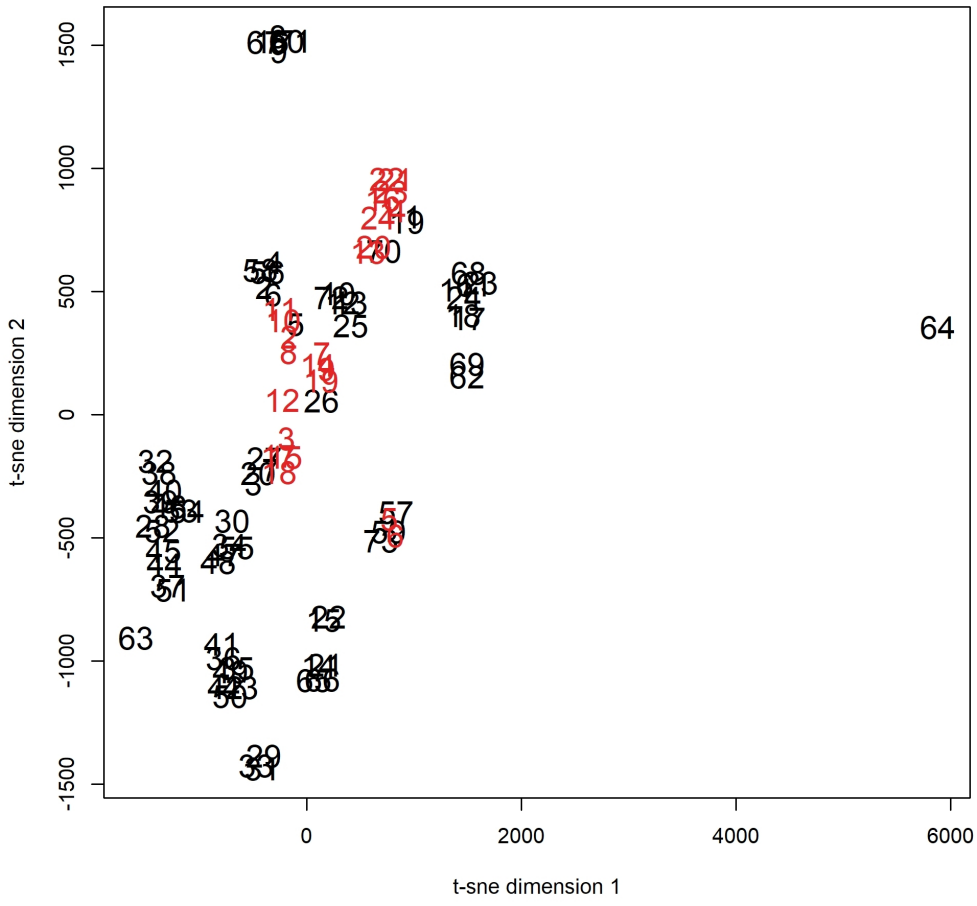

Figure 4.39: t-sne visualization of CEC vs BBOB two-dimensional problems, using all CEC problems.

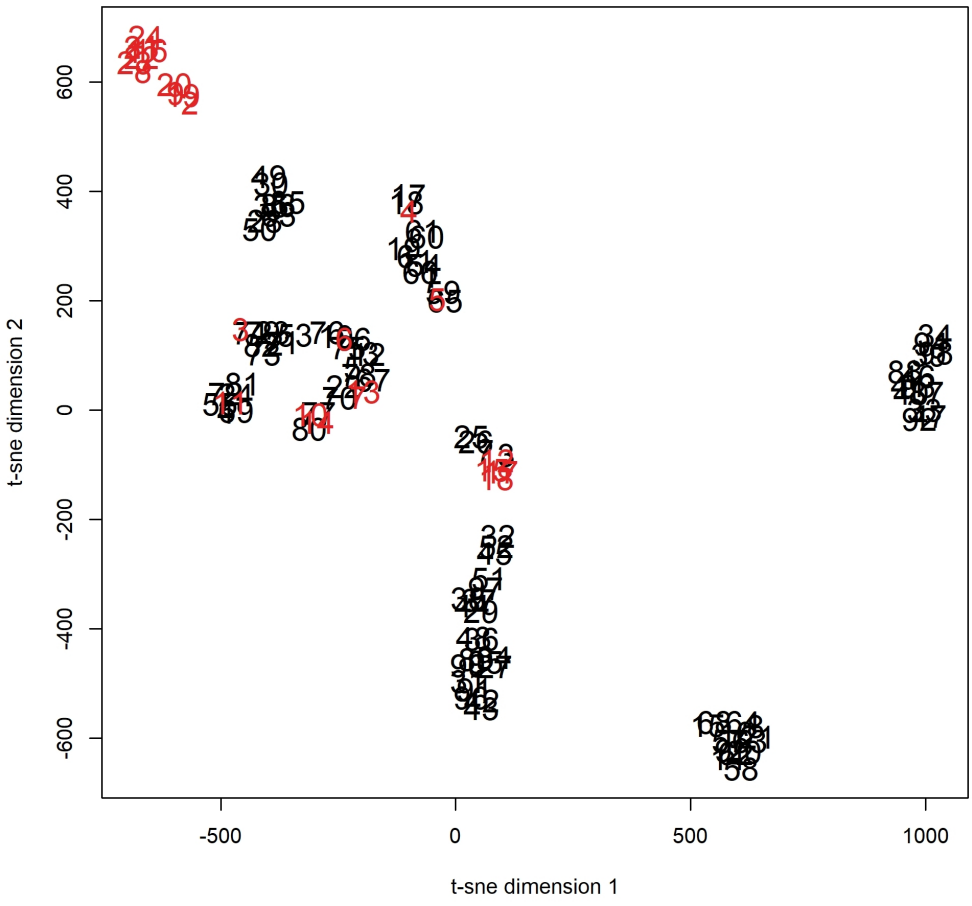

Figure 4.40: t-sne visualization of CEC vs BBOB 10-dimensional problems, using all CEC problems.

Table 4.11: Common problems between the CEC 2014 and BBOB problem sets.

Problem id CEC	Problem id BBOB	Problem name
2	12	Bent Cigar Function
3	11	Discus Function
4	9	Rosenbrock's Function
6	16	Weierstrass Function
8	3	Rastrigin's Function
10	20	Schwefel's Function
12	23	Katsuura Function

4.3 Discussion and Conclusion

In this chapter, we have presented a study of the generalizability of ELA between different problem sets. We have shown that a large number of landscape features are not invariant to the transformations of shifting and scaling of the problem samples and that this non-invariance makes it difficult to generalize the information about problem landscapes between different problem sets. This is because even though different problem sets might contain problems that are similar to one another based on their mathematical definitions, these similar problems are often transformed, and these transformations will make the problems appear entirely different from one another if they are analyzed using ELA.

In the first part of this chapter, we conducted a comprehensive study on the effects of sampling strategy, sample size, dimension, and problem set on the invariance of landscape features. There are several important points that are raised by the findings of these experiments. First, we show that a large number of landscape features are not invariant to the transformations of shifting and scaling of the problem samples used to calculate the landscape features. This shows the importance of understanding the landscape features that are used when we perform our experiments. For example, if we are interested in algorithm performance, we need to be aware that some landscape features can change even when performing transformations that we expect would not affect algorithm performance. While the flacco library makes it easy to calculate every landscape feature, we still need to be careful and only select those features that are relevant to our experiments. Even though modern machine learning techniques might be able to automatically eliminate the features which are not invariant, limiting the number of features used can still bring performance improvements, both in the training time of the machine learning algorithms and in the time needed to calculate the features. Further, it is possible that some of these features could still deceive the machine learning algorithm, or lead to overfitting because they only separate the prediction classes in the training set and not in the testing set.

Second, we show that the choices of sample size, sampling strategy, dimension, and problem set all have an effect on this invariance. Renau et al. (Renau et al., 2020) have already shown that there is a difference in landscape features when using different sampling methods. But we have now shown that the choice of sampling method also has an effect on the invariance of landscape features. In other words, simply starting from different samples has an effect on the landscape features produced when these samples are either scaled or shifted. This even further shows that the type of sampling has to be taken into account when analyzing landscape features. To our knowledge, a large amount of current work on landscape features has been performed mainly using random uniform and Latin hypercube sampling (Renau et al., 2020). If the goal of ELA is to produce more robust features, we believe that our work provides further evidence that different sampling types

Table 4.12: BBOB problems.

Problem id	Problem name
1	Sphere Function
2	Ellipsoidal Function
3	Rastrigin's Function
4	Büche-Rastrigin's Function
5	Linear Slope
6	Attractive Sector Function
7	Step Ellipsoidal Function
8	Rosenbrock's Function, original
9	Rosenbrock's Function, rotated
10	Ellipsoidal Function 2
11	Discus Function
12	Bent Cigar Function
13	Sharp Ridge Function
14	Different Powers Function
15	Rastrigin's Function
16	Weierstrass Function
17	Schaffer's F7 Function
18	Schaffer's F7 Functions, moderately ill-conditioned
19	Composite Griewank-Rosenbrock Function F8F2
20	Schwefel's Function
21	Gallagher's Gaussian 101-me Peaks Function
22	Gallagher's Gaussian 21-hi Peaks Function
23	Katsuura Function
24	Lunacek bi-Rastrigin Function

should be used when evaluating landscape features.

In the second part of this chapter, we have demonstrated the effect of landscape feature invariance by conducting a complementarity analysis experiment that examined the effectiveness of ELA-based problem representation when comparing problems from different benchmark sets. We have shown that using all available landscape features produces a very poor representation because the shifting and scaling of the samples splits the problem sets into two distinct groups. As we have seen in the previous chapters, such transformations, when applied to the different instances of the BBOB problem set, do not have an effect on algorithm performance. Because of this, removing the features that are not invariant to shifting and scaling is necessary in order to create a good representation. However, this approach still has limitations, and some problems which should be similar to one another based on their problem definitions are instead treated in the representation as being different from one another. This could be due to the fact that the benchmarks analyzed perform the transformations of shifting and scaling on the underlying problem rather than on the samples as we have done in our experiments, or because the benchmarks include additional transformations such as rotation.

Further work needs to be done to determine whether these differences are actually practically significant. That is, whether they affect algorithm performance. Additionally, our analysis did not account for problem transformations that act upon the underlying search space, rather than on the problem samples. In our experiments, we performed the transformations of shifting and scaling by transforming the problem samples directly, meaning

that the area of the problem on which the samples were originally collected remained the same regardless of the transformation performed. However, when these transformations are performed by optimization benchmarks, they are performed on the underlying problem before it is sampled. This can potentially change the area of the search space that the problem is sampled from. This could explain why our complementarity analysis was not able to correctly explain all of the problems.

Chapter 5

Combining the Performance and the Problem Space

In the previous two chapters, we have examined the performance and the problem spaces of optimization separately. In Chapter 3, which dealt with the performance space, we have shown that algorithm performance is dependent on the problem being solved and that years of optimization benchmarking competitions have not resulted in an algorithm that would perform well on a varied set of problems. Because of this, automated algorithm selection, or the ability to automatically select an algorithm that will perform well on a specific problem, is crucial for achieving good algorithm performance. In order to improve algorithm selection results, we analyzed the problem space of optimization in Chapter 4. Specifically, we examined the performance of exploratory landscape analysis and showed that while ELA has produced good results when used for algorithm selection in the past, it appears to have difficulties generalizing between different sets of problems.

In this chapter, we combine the perspectives of both the performance and the problem spaces by performing algorithm selection, and by examining the generalizability of ELA when used for this purpose. In the first part of the chapter, we use machine learning to create several automated algorithm selection models that are trained and tested on two different problem sets. In order to further examine the generalizability of ELA, we also explore the use of artificial problem generators by having one of our problem sets consist of artificially generated problems. This helps increase the variety in the dataset used to train the algorithm selection model compared to using just the handmade benchmark problems that we have used so far. In this chapter, we evaluate one such problem generator by using its problems as training data for our algorithm selection model.

In the second part of this chapter, we perform experiments to investigate and explain the performance of our algorithm selection model. We do this in two ways. First by investigating the correlation between different sets of problems, which allows us to see if the problems in the two sets are similar, or if they form two distinct groups. And second by using a framework called Shapley Additive Explanations (SHAP) to calculate feature importance of our machine learning models, explain why our models made the decisions they did, and determine if there are certain landscape features that deceive some of the models and prevent them from making correct decisions. The source code for the experiments performed in this Chapter is available at https://github.com/UrbanSkv/ela-transfer-learning.

5.1 Methodology

The methodology used in this chapter is split into three parts: the problem portfolio representation part, the supervised machine learning part, and the complementarity analysis part.

In the problem portfolio representation part, we introduce the two datasets used for training and testing the automated algorithm selection models, with one set containing artificially generated problems and the other containing handmade benchmark problems. We then create the problem portfolio representation by calculating the landscape features that will be used as training attributes in the supervised machine learning part of our methodology.

In the supervised machine learning part, we examine the performance of a machine learning algorithm selection model on the two problem sets introduced in the portfolio representation part. In particular, we are interested in seeing how an algorithm selection model trained on a set of artificially generated problems behaves when tested on the set of handmade benchmark problems, i.e., how well the model can generalize the information obtained from the training set.

In the complementarity analysis part, we further examine the complementarity of the artificially generated problem set with the handmade problem set by using the methodology from (Eftimov, Popovski, et al., 2020), which uses Pearson correlation in combination with Singular Vector Decomposition to better detect problem correlation and by using SHAP values to determine feature importance of the algorithm selection models.

5.1.1 Problem Portfolio Representation

The first step of our methodology involves generating an artificial benchmark problem set, selecting the handmade problems that we will use for our comparison, and calculating landscape features.

The artificial problem set is generated using the generator described by Tian et al. in (Tian et al., 2020). The generator works by representing an optimization problem as a tree, which is randomly constructed using a fixed set of operators. In addition to traditional mathematical operators such as multiplication and division, the generator also uses 8 different difficulty injection operations which are designed to introduce difficulties commonly encountered in real-life problems, such as adding noise or introducing multimodality. After the tree is grown to a predefined size, it is converted into a reverse Polish notation so that it can be more easily evaluated.

The best-performing algorithm on a generated problem is determined by running 10 optimization algorithms on the problem and selecting the best-performing one. The problem generator is designed to generate problems that are only solved by a single best-performing algorithm. After the mean objective values are computed for each of the 10 algorithms, a Wilcoxon sum-rank test is performed to determine if there are any algorithms for which there is no statistically significant difference in performance when compared to the best-performing algorithm. If such algorithms exist, the generated problem is discarded. This ensures that for every generated problem, there is only a single best-performing algorithm from a statistical point of view. Table 5.1 lists the 10 algorithms used. Table 5.2 lists the parameters used for each algorithm.

Other than the population size, which is set at 100 for every algorithm, the parameters use default values from the literature in which they were introduced. Each algorithm is evaluated using 20 independent runs, and the mean of the minimum objective value is used as the performance metric. The termination condition of each run is set at $1000D$ function evaluations, with $D = 10$. A brief description of the algorithms used is provided below.

Table 5.1: The algorithms used by the artificial problem generator.

Name	Abbreviation	Reference
Artificial Bee Colony	ABC	2005
Ant Colony Optimization	ACO	2008
Competitive Swarm Optimizer	CSO	2014
Covariance Matrix Adaptation Evolution Strategy	CMA-ES	2001
Differential Evolution	DE	1997
Fast Evolutionary Programming	FEP	1999
Genetic Algorithm	GA	1996
Particle Swarm Optimization	PSO	1995
Simulated Annealing	SA	1987
Random Search	Rand	2010

Table 5.2: The parameters used by the algorithms of the artificial problem generator.

Algorithm	Parameters
ABC	$swarm\_size = 100$, $limit = num\_onlookers * dim$ $num\_onlookers = 0.5 * swarm\_size$ $num\_employed\_bees = 0.5 * swarm\_size$ $num\_scouts = 1$
ACO	$ants\ used\ in\ an\ iteration = 100$ $q = 0.3$ $\xi = 0.8$
CMA-ES	$\lambda = 100$ $\mu = \lfloor \lambda/2 \rfloor$ $\omega_i = ln(\frac{\lambda+1}{2}) - ln(i)$ $C_c = \frac{4}{n+4}$ $C_{cov} = \frac{2}{(n+\sqrt{2})^2}$ $C_\sigma = \frac{4}{n+4}$ $d_\sigma = C_\sigma^{-1} + 1$
CSO	Acceleration constant = 1
DE	$F = 0.5$, $CR = 0.1$, $strategy = \text{rand/1/bin}$
FEP	$tournament\ size = 10\ \eta = 3$
GA	$T\_max = 200$ $Crossover\ probability = 1.0$ $U_m = 0.5$
PSO	$Acceleration\ constant = 1$ $Inertia\ weight = 0.4$
SA	$Initial\ temperature = 0.1$ $Cooling\ factor = 0.99$ $\sigma = 0.98$
Rand	No parameters

Artificial Bee Colony (Karaboga, 2005) is a swarm optimization algorithm inspired by the behavior of bees that splits the candidate solutions of its population into three types: scouts, onlookers, and employed bees. In every iteration, the candidate solutions are updated based on their type. The scout bees provide the algorithm with exploration power, while the other two types provide exploitation power.

Ant Colony Optimization (Socha & Dorigo, 2008) is a swarm optimization algorithm inspired by the behavior of ants. In every iteration, the candidate solutions are updated based on a trail of artificial pheromones that are left behind by every candidate solution.

Covariance matrix adaptation evolution strategy (Hansen & Ostermeier, 2001) is an optimization method similar to quasi-Newtonian methods that aim to approximate the second-order derivative matrix of the underlying problem. Unlike the quasi-Newtonian methods, it does not use gradients, which makes it usable on a larger amount of problems. In every iteration, the candidate solutions are updated by sampling from a multivariate normal distribution, which is represented by a covariance matrix and updated based on the quality of the found solutions, random perturbations, and the history of the search process.

Differential Evolution (Storn & Price, 1997) is an evolutionary algorithm designed for numerical optimization. In every iteration, the candidate solutions are updated using two simple mathematical formulas for crossover and mutation.

Fast Evolutionary Programming (Yao et al., 1999) is a modification of a traditional evolutionary algorithm, where in every iteration the candidate solutions are updated using only mutation based on a statistical distribution that is unique to each candidate solution. Unlike traditional evolutionary programming, fast evolutionary programming uses the Cauchy rather than the Gaussian distribution for the mutation operator in order to improve the speed of convergence, as the Cauchy distribution produces larger jumps more often, increasing the algorithm's exploration power at the expense of its exploitation power.

Genetic Algorithm (Deb & Goyal, 1996) is a traditional genetic algorithm, where candidate solutions are represented as a series of bits. In every iteration, the candidate solutions are updated using mutation (flipping random bits) and crossover (combining the bits of two candidate solutions). While genetic algorithms were initially designed to solve combinatorial problems, the implementation used by the artificial generator uses a method that simulates binary crossover for continuous domains.

Particle Swarm Optimization (Eberhart & Kennedy, 1995) is a swarm optimization algorithm where in every iteration each candidate solution is updated by its own best-known position, as well as the global best-known position.

Competitive Swarm Optimizer (Cheng & Jin, 2014) is a modified Particle Swarm Optimization algorithm where the candidate solutions are no longer updated based on best-known positions, but by a competition system that pairs two candidate solutions together, with the losing solution (the solution with the lower fitness) updating its position based on the winning solution (the solution with the higher fitness).

Simulated annealing (Van Laarhoven & Aarts, 1987) traditionally uses a single candidate solution. In every iteration, the solution either stays in its current position or switches to the new point based on the evaluation of the new point as well as

on a probability function. The candidate solution will always switch to a solution that is better than the current one but also has a chance to switch to a worse solution. The probability of swapping to a worse solution decreases with time, making the algorithm shift from exploration to exploitation. The problem generator uses a population-based variant, which evaluates multiple candidate solutions in each iteration rather than using just a single candidate solution.

Random Search (Zabinsky, 2010) is the most primitive optimization algorithm. In every iteration, each candidate solution is assigned a random position in the search space.

The handmade benchmark problem set consists of the problems from the BBOB problem set (Hansen et al., 2012), which we have examined in the previous chapters. In this experiment, we use the first 15 instances of each of the 24 base problems, for a total of 360 instances. We will refer to the 24 base problems as the base BBOB problems, and the individual instances as BBOB instances. We determine the best-performing algorithms on the BBOB instances using the same procedure with the same set of 10 optimization algorithms as used for the artificial instances. However, we do not discard problems where multiple algorithms achieve performance that is not statistically significantly different from the best-performing algorithm.

In total, our final machine learning dataset contains 500 artificial instances, 50 for each algorithm, as well as 360 BBOB instances. For both the artificial and the BBOB instances, we collect $250D$ instance samples. We set $D = 10$ to remain consistent with (Tian et al., 2020). These samples are then used to calculate landscape features in the same manner as in the previous chapter. These landscape features are used as a basis for all following experiments.

5.1.2 Machine Learning

In the second part of our methodology, we train and evaluate an algorithm selection model to determine the best-performing algorithm on the instances of the two problem sets described in the previous chapter, using the landscape feature representations introduced in the previous section and the random forest machine learning algorithm (Breiman, 2001).

The dataset used for machine learning consists of landscape features for the attributes, and the best-performing algorithm on the specific instance as the prediction class. To account for the BBOB instances that contain multiple algorithms for which there is no statistical difference in performance, two types of machine learning experiments are performed. In the first, only the single absolute best-performing algorithm is used as a prediction class. In the second, which we will call the multi-label scenario, the prediction of the model is considered correct if any one of the algorithms for which there is no statistical difference compared to the absolute best-performing algorithm is predicted. The second type mirrors a practical application of algorithm selection, where we are not interested in finding all possible algorithms that solve a particular problem well, and are instead satisfied with finding a single well-performing algorithm.

Another way of performing algorithm selection would be by performing regression instead of classification, by predicting not simply the best performing class, but rather the performance of each algorithm on each problem. In our experiments, we chose to use classification to remain consistent with the methodology that was used by Tian et al. in the original evaluation of the artificial problem generator (Tian et al., 2020)

In order to evaluate machine learning performance, we define and train five different algorithm selection models that differ in the data used for the training and the testing sets. These models are:

Instance Split BBOB The model is trained on the BBOB instances and tested on the BBOB instances, using instance-based stratified cross-validation. Here, the training data contains 14 instances of each of the 24 base problems, and the testing data contains the final 15^{th} instance of each base problem. This procedure is repeated 15 times, once for each instance. This means the model can learn from all of the 15 instances.

Problem Split BBOB The model is trained on the BBOB instances and tested on the BBOB instances, using problem-based stratified cross-validation. Here, the training data contains all 15 instances of 23 base problems, and the testing data contains the 15 instances of the remaining base problem. As in the instance split model, this is repeated 24 times, once for each base problem. This means the model only learns on 23 base problems and then predicts the remaining base problem with no information about this problem contained in the testing set. We expect this to produce poorer results compared to the instance split model, as the training set contains no information about the base problem used for the testing data.

Artificial The model is trained on the artificial instances and tested on the artificial instances. Since the artificial set cannot be split by base problems, we split the training and testing sets randomly using 10 fold cross-validation.

Combined The model is trained on a combined set of both artificial and BBOB instances and tested on the combined set of instances. 10 fold cross-validation is used to split the training and testing set.

Generalized The model trained on the artificial instances and tested on the BBOB instances. The full artificial set is used for training, and the full BBOB set is used for testing. We chose to use the artificial instances as the training set to ensure that the training set contained a balanced set of classes, and because some classes present in the artificial set are not present in the BBOB set.

With the first three models, we are interested in seeing how the models perform on both of the problem sets in isolation, to obtain a baseline performance metric. With the final two models, we focus on the main task of this chapter, first by combining the two problem sets, and finally by examining whether or not the knowledge obtained from the artificial instances can be generalized to the BBOB instances.

The artificial model is trained and tested on the instances from the artificial set, with 10-fold cross-validation used to achieve more reliable results. For the combined model, both the artificial and the BBOB instances are combined into a single set that is then used for both training and testing, also using 10-fold cross-validation. Finally, the generalized model is trained on the artificial instances and then tested on the BBOB instances. Here, no cross-validation is used, as we already use different sets for training and testing. However, since we use the random forest algorithm to perform machine learning, we repeat this experiment 10 times and use mean values of the performance metrics to obtain more reliable results.

5.1.3 Complementarity Analysis

Our complementarity analysis methodology is split into two parts. In the first part, we examine the complementarity of the artificial and the BBOB problem sets using correlation analysis and Singular Value Decomposition (SVD). In the second part, we take a deeper look into the feature importance of the different machine learning models, focusing on the generalized model and on the model that was trained and tested on the instance split BBOB model.

In the first part of our complementarity experiments, we plot the correlation between the problems of the two problem sets. To improve the result of this analysis, we use SVD, as described in (Eftimov, Popovski, et al., 2020). Singular value decomposition is a mathematical procedure that is used in machine learning to transform the original data, represented as a matrix M, into $M = U\Sigma V^T$, where Σ is a diagonal matrix that contains singular values on its diagonal. These singular values provide us with a new representation of the data. This is beneficial because the singular values are not linearly correlated with one another. As we have shown in the previous chapter, a large number of the landscape features used in our initial dataset are correlated. Because of this, transforming the landscape features in this way has been shown to improve the quality of the visualization (Eftimov, Popovski, et al., 2020). We perform the SVD projection by first calculating the SVD mapping using the landscape features from the artificial instances, and then projecting the BBOB features into the SVD subspace of the artificial features. This is done to avoid biasing the training set with the information from the testing set.

In the second part of the complementarity analysis, we examine the results of our machine learning models using a technique called *Shapley Additive Explanations* (SHAP). This technique produces for every prediction of a machine learning model a set of SHAP values which show how each of the features of the model (in our case, the ELA features) affected its decision. The SHAP values are based on the concept of Shapley values, originating from game theory. Shapley values explain the contribution of each player in a team to the overall result of the game. SHAP values use this concept for model explanation, substituting players for model attributes, and the result of a game for a prediction of the model. One drawback of Shapley values is that they are computationally expensive to compute, as they require calculations to be performed for every possible combination of players. Fortunately, multiple algorithms exist that can reduce its computational complexity. Kernel SHAP (Lundberg & Lee, 2017) can be used to calculate approximate SHAP values in a model agnostic way, while the Tree SHAP (Lundberg et al., 2020) algorithm can find exact SHAP values in polynomial time but is limited to tree-based models. Since we are using tree-based models, we use the Tree SHAP algorithm implemented in the Python library shap (Lundberg, 2018) to calculate the SHAP values using the default parameter values provided by the library. Similar analysis has already been successfully used when examining the use of ELA features in machine learning, for example in (Nikolikj et al., 2022; Trajanov et al., 2021). However, these studies focused on regression learning. In our case of using a multiclass classification model, we obtain SHAP values not just for the class that the model predicted, but for every one of our 10 classes (algorithms). When interpreting the SHAP values produced in the multiclass scenario, for every individual prediction, the sum of the SHAP values represents the difference between the probability of that prediction being made by the model and the baseline probability, with the baseline probability equal to 0.1 due to our dataset containing 10 classes.

5.2 Results

In this section, we present the results of the experiments presented in the previous section. First, we conduct preliminary experiments in order to determine the distribution of the artificially generated problems and to determine if a small sample size of the artificial problems still provides results representative of the ones achieved in the paper that introduced the problem generator (Tian et al., 2020). Then, we show the results of the algorithm selection experiment, showing the results of all of the algorithm selection models that were used for our experiments.

5.2.1 Problem Selection and Feature Calculation

Before proceeding with the artificial problem generation, we first wanted to verify that the algorithm generation procedure would produce the same distribution of problems as the one described in (Tian et al., 2020), even when using a smaller sample size. As the process of calculating landscape features is time-consuming, we had to use a smaller number of problems than used in (Tian et al., 2020), where exploratory landscape analysis was not used. Because of this we first wanted to ensure that a smaller number of problems would not introduce additional bias. The artificial problem generation works as described in (Tian et al., 2020), using the source code that was made available online[1] by the authors of the paper.

Figure 5.1 shows an example distribution of 500 problems generated in a single run, based on which optimization algorithm found the best final solution on the generated problems, using the algorithms and algorithm parameters described in (Tian et al., 2020). We can see that this distribution broadly matches the one in the original algorithm generation paper, with GA (Deb & Goyal, 1996) and CMA-ES (Hansen & Ostermeier, 2001) algorithms solving the majority of the generated problems.

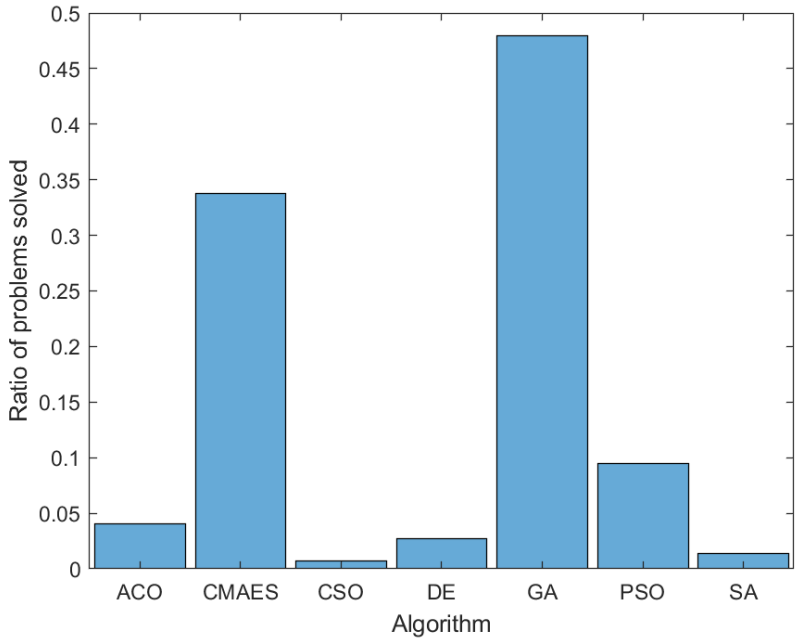

Figure 5.1: A sample distribution of generated problems, based on which optimization algorithm best solves the generated problem.

For a more thorough evaluation, we conducted a statistical analysis where different sample sizes were tested based on the algorithm frequency of solved problems, as we wanted to find the smallest number of problems to generate that would preserve the problem distribution from (Tian et al., 2020). We generated multiple different sets of problems, with the number of problems (i.e., sample size) in $\{500, 1000, 5000, 10000\}$. For each sample size, we repeated the sampling 10 times. Then, the samples were compared using the information of how many problems were solved by each of the 10 algorithms. For this purpose, we used the Friedman test, since we were comparing more than two samples and the required conditions for the safe use of the parametric tests were not satisfied. For

[1]The source code is available at https://github.com/BIMK/Algorithm-Recommendation

comparisons between different sample sizes, all their samples were compared, for a total of 20 samples, 10 for each sample size. Table 5.3 shows the p-values obtained for the comparisons. We can see that all p-values are above the critical value of $\alpha = 0.05$, and thus there is no statistically significant difference between the samples regardless of the sample size used. The sample size of 500 was the lowest one we evaluated, as it was small enough to be computable in a reasonable amount of time, but still large enough to be roughly comparable to the BBOB dataset. This shows that even under a smaller sample size, the problems produced by the generator are still representative of the ones in (Tian et al., 2020).

Table 5.3: P-values of the Friedman test used to determine whether multiple runs of the problem generation algorithm produce the same distribution of problems depending on the number of problems used.

Number of problems	500	1000	5000	10000
500	0.9995	0.9998	0.9715	0.9987
1000	-	0.9926	0.9922	0.9987
5000	-	-	0.9973	0.9989
10000	-	-	-	0.9302

5.2.2 Algorithm Selection

For the algorithm selection analysis, we create two separate datasets. The first consists of artificial benchmark instances, while the second consists of handmade BBOB instances. The artificial problem set contains 500 generated instances, with 50 instances from each of the 10 different prediction classes (algorithms that best solve the problem). This was done in order to create a balanced machine-learning dataset. The BBOB problem set contains the first 15 instances of each of the 24 base problems, for a total of 360 instances.

To obtain a more robust set of features, each instance is sampled separately 10 different times, using Latin hypercube sampling with a sample size of $250D$, the dimension D of 10, and taking samples from the range of $[-5, 5]$. A single dimension of $D = 10$ was chosen because it offers a balance between problem complexity and computational costs, and because this is the same dimension that was used in the paper that developed the problem generator used in our experiments (Tian et al., 2020). As the complexity of calculating some landscape features raises polynomially or with regards to the dimension and the number of samples, higher dimensions are much harder to compute. We calculate the landscape features separately for each of these samplings, creating 10 different sets of landscape features for each problem. The median of these 10 sets is used as the final set of landscape features. The use of medians to obtain a more robust set of features has been used for example in (Jankovic, Eftimov, et al., 2021; Jankovic, Popovski, et al., 2021).

The landscape features were calculated using the R library flacco. We used the landscape feature categories *cm_angle*, *cm_grad*, *ela_level*, *ela_meta*, *ic*, *disp*, *limo*, *nbc*, and *pca*. As in the previous chapter, this excludes some flacco feature categories, particularly those that are either computationally expensive in higher dimensions or require an exact problem definition to be known and additional sampling to compute.

Some of the landscape features produced errors or resulted in NA values when computed on certain artificially generated instances. As the number of instances is limited, we chose to remove these landscape features from our final machine-learning dataset to ensure that all of the artificial instances could be used for the machine-learning data. The full list of 44 landscape features used is listed in Table 5.4.

Table 5.4: All landscape features used as machine learning attributes, with the names taken from the flacco library.

cm_angle.dist_ctr2best.mean	cm_angle.dist_ctr2worst.mean
cm_angle.angle.mean	cm_grad.mean
ela_meta.lin_simple.adj_r2	ela_meta.lin_simple.intercept
ela_meta.lin_simple.coef.min	ela_meta.lin_simple.coef.max
ela_meta.lin_simple.coef.max_by_min	ela_meta.lin_w_interact.adj_r2
ela_meta.quad_simple.adj_r2	ela_meta.quad_simple.cond
ela_meta.quad_w_interact.adj_r2	ic.h.max
ic.eps.max	ic.m0
disp.ratio_mean_02	disp.ratio_mean_05
disp.ratio_mean_10	disp.ratio_mean_25
disp.ratio_median_02	disp.ratio_median_05
disp.ratio_median_10	disp.ratio_median_25
disp.diff_mean_02	disp.diff_mean_05
disp.diff_mean_10	disp.diff_mean_25
disp.diff_median_02	disp.diff_median_05
disp.diff_median_10	disp.diff_median_25
limo.avg_length.reg	limo.length.mean
limo.ratio.mean	nbc.nn_nb.sd_ratio
nbc.nn_nb.mean_ratio	nbc.nn_nb.cor
nbc.dist_ratio.coeff_var	nbc.nb_fitness.cor
pca.expl_var.cov_init	pca.expl_var.cor_init
pca.expl_var_PC1.cov_init	pca.expl_var_PC1.cor_init

All machine learning is performed using the random forest (Breiman, 2001) algorithm, as this algorithm has been shown to perform well when predicting algorithm performance using the BBOB dataset (Kerschke & Trautmann, 2019a). The random forest algorithm is an upgrade to the traditional decision tree. Decision trees build their model by creating a tree that starts at a root node and then splits the dataset according to a single attribute in each node. However, such models often overfit the training data if techniques such as pruning are not used (Ying, 2019). The random forest algorithm improves on this shortcoming by using an ensemble of simple decision trees, with each tree using only a randomly selected subset of all available attributes in each split and a random sample of all available instances in each tree. Similar to bootstrap sampling, this decreases the variance of the model without increasing its bias. We use the R library randomForest (Liaw & Wiener, 2002) which provides an implementation of the algorithm in the R programming language. We use the parameter $ntree = 1000$, and the default parameters values for all other parameters. The number of trees was determined experimentally. Generally speaking, increasing the number of trees increases the performance of the algorithm but requires additional computational time. However, past a certain point, additional trees no longer produce a difference in performance. We determined that increasing the number of trees past 1000 did not produce any improvement in results. The full parameters used are listed in Table 5.5.

Table 5.5: Full list of parameters used for the random forest algorithm. Other than the number of trees which is set to 1000, all other parameters use default values. The left side lists the parameter name, with the name used in the R code specified in brackets.

Parameter	Value
Number of Trees (ntree)	1000
Number of sampled variables (mtry)	sqrt(*number of variables*)
Sampling with replacement (replace)	True
Cutoff (cutoff)	$\frac{1}{number\ of\ classes}$
Sample Size (sampsize)	*Number of instances*
Minimum size of terminal nodes (nodesize)	1
Maximum number of terminal nodes (maxnodes)	No limit

The number of sampled variables determines how many random attributes are considered for each split. The square root of the number of classes is a typical default value (Friedman et al., 2001) for classification. The sampling with replacement parameter determines if the instances sampled for each decision tree should be done with replacement or without. Sampling with replacement is generally done, as it reduces the variance of the model without increasing its bias, in the same way as bootstrap sampling. The cutoff parameter determines how the ensemble of trees selects a single winner and essentially allows us to specify weights for each of the prediction classes so that certain classes require a higher number of votes to be selected. Leaving this parameter at $\frac{1}{number\ of\ classes}$ assigns equal weights to all classes. Finally, the minimum size of terminal nodes and the maximum number of terminal nodes can be used to limit the size of the generated trees, resulting in faster performance at the expense of limiting the types of trees that the model can create. The default values of 1 and unlimited put no limits on the types of trees that can be generated, and are used as our dataset is small enough that it does not require the additional performance benefits of smaller trees.

Before analyzing the results of the automated algorithm selection model, we will first more closely examine the data used to make the predictions. Table 5.6 shows which algo-

rithms achieved the absolute best optimization result for every BBOB base problem after $1000D$ function evaluations. However, each base problem contains 15 distinct instances, and these instances can be solved best by different optimization algorithms. Because of this, the table can show several algorithms as the best-performing algorithm for each base problem. We believe that this occurs due to the stochastic nature of the optimization algorithms. As we have seen in Chapter 3, there is only a small, not statistically significant, difference between some algorithms on the BBOB problem set. However, our experimental setup in the current chapter does not account for statistical significance, and only selects the absolute best-performing algorithm. Because of this, it is possible that even very small differences in algorithm performance could result in selecting a different best algorithm between multiple instances of the same base problem, despite the fact that different instances should not affect algorithm when considering statistical significance.

Table 5.6: The algorithms that found the best final solution on each individual BBOB base problem. In some cases, certain BBOB instances belonging to the base problem were solved by a different algorithm. For the problems where this occurred, the other algorithms are listed in brackets.

Problem id.	Best Algorithm
1	CMA-ES (GA)
2	ACO
3	GA
4	GA
5	CMA-ES
6	GA
7	DE (CSO, ABC, CMA-ES)
8	CMA-ES(ACO, GA, CSO)
9	CSO (CMA-ES)
10	CSO
11	DE (CSO)
12	GA
13	GA (CSO, CMA-ES)
14	CMA-ES (GA, CSO)
15	CSO
16	CSO
17	CSO (GA, CMA-ES, ACO)
18	CSO (GA, CMA-ES)
19	PSO (CSO, GA)
20	GA
21	FEP (CSO, GA, CMA-ES)
22	FEP (ACO, CSO, GA, ABC)
23	PSO (GA)
24	GA (CSO)

A large difference between the BBOB and the artificial instances is that for the BBOB instances, it is possible that multiple algorithms are statistically tied as the best-performing algorithm. Specifically, only 140 out of 360 instances contain a single best-performing algorithm, while 220 instances contain multiple algorithms for which there is no statistical difference in performance when compared to the best-performing algorithm.

We deliberately chose not to modify the original artificial problem generator so that it

would not discard instances where there was no statistical difference in the performance of multiple algorithms. This was done because we wanted to limit the scope of our experiments to only use the original problem generator exactly as described in (Tian et al., 2020) without any modification and because we did not want to modify the artificial problem generation procedure using the knowledge of the BBOB problem set. We also chose to use the entire set of 360 BBOB instances, rather than just the set of 140 instances that contain a single best-performing algorithm, as we did not want to tailor the composition of the BBOB set based on the knowledge of how the artificial set was constructed.

This difference in the performance of the algorithms indicates a potentially large difference between the instances of the artificial and the BBOB sets. The artificial set only contains a relatively small subset of all possible instances that could be generated by the artificial problem generator, as only about 30% of the instances contain a single best-performing algorithm. On the other hand, the majority of BBOB instances have multiple algorithms for which there is no statistical difference in performance. Because of this, the only potential for overlap between the two problem sets are the 140 BBOB instances that are only solved well by a single algorithm.

Table 5.7 shows the accuracy, as well as the macro precision and recall scores for the five algorithm selection models described in Section 5.1 when all landscape features were used and only the single best performing algorithm was used as a classification target for the BBOB instances. Accuracy is defined as $\frac{TP+TN}{TP+FP+TN+FN}$, precision as $\frac{TP}{TP+FP}$, and recall as $\frac{TP}{TP+FN}$, where TP, TN, FP, and FN are true positives, true negatives, false positives, and false negatives.

Table 5.7: The results of the algorithm selection model trained using every available landscape feature.

Model	Accuracy	Precision	Recall
BBOB (Instance Split)	0.68	0.69	0.60
BBOB (Problem Split)	0.21	0.24	0.32
Artificial	0.54	0.52	0.54
Combined	0.61	0.60	0.59
Generalized	0.20	0.13	0.14

The macro precision and recall metrics are calculated by taking the mean of the per-class precision and recall. In some cases, it is possible for the precision and recall to be undefined for some classes. Precision is undefined if both true positives and false positives equal 0 (i.e., no instance in the testing set was classified with a specific class). Recall is undefined if both true positives and false negatives are 0 (i.e., there is no instance of a specific class in the testing set). If the values for precision or recall are undefined for a specific class, this class is not used when calculating the macro metrics.

In the models where cross-fold validation is used, the final precision and recall metrics are the mean values across all folds. In the generalized model, when no cross-fold validation is used but the experiments are repeated 10 times to achieve more robust results, the mean of the 10 runs is used.

We can see that the choice of the train-test split has a large effect on the results of the two models trained on BBOB data. If the training and testing sets are stratified per instance, the model achieves an accuracy of 0.68. However, if the sets are stratified per problem, the accuracy drops to 0.26. These results show the importance of having similar instances in the training and in the testing set. The artificial and the combined set achieve an accuracy of 0.52 and 0.61, respectively.

Finally, generalized learning produces poor results, with an accuracy of only 0.20. This indicates that the two problem sets are distinct from one another and that the instances in the artificial set do not contain enough information about the instances in the BBOB set, similar to the results of when the BBOB set is stratified per problem. The precision and recall scores likewise indicate very poor results.

For a more detailed examination of the generalized learning results, Table 5.8 shows the confusion matrix of the predictions. We can see that the predictions of the model are heavily concentrated on a few of the classes, specifically CMA-ES, CSO, and DE.

Table 5.8: The confusion matrix for generalized learning, using all landscape features.

Pred. \ True	ABC	ACO	CMA-ES	CSO	DE	FEP	GA	PSO	SA	Rand
ABC	0	0	0	0	0	0	0	0	0	0
ACO	0	2	2	5	0	3	3	2	0	0
CMA-ES	0	2	10	12	12	0	43	11	0	0
CSO	1	12	17	47	2	9	44	1	0	0
DE	1	4	9	27	6	1	16	8	0	0
FEP	0	0	0	0	0	0	0	0	0	0
GA	0	0	0	0	0	0	9	0	0	0
PSO	0	4	8	16	0	0	9	0	0	0
SA	0	0	0	1	0	0	0	0	0	0
Rand	0	0	0	0	0	0	1	0	0	0

In order to attempt to improve generalized learning accuracy, we used the findings from Chapter 4 and limited the machine learning data to only the features that were found invariant to the transforms of shifting and scaling. The results from these experiments are presented in Table 5.9, and the specific landscape features used are presented in Table 5.10. We only used a set of 13 invariant landscape features, as some of the features that were found invariant in Chapter 4 were removed from our data as they produced errors when calculated on some of the artificially generated problems. We can see that this produces similar or even slightly better results, which indicates that these 13 features alone contain a similar amount of information relevant for classification compared to the full 67 feature set.

Table 5.9: The results of the algorithm selection model trained using invariant features.

Model	Accuracy	Precision	Recall
BBOB (Instance Split)	0.68	0.68	0.62
BBOB (Problem Split)	0.26	0.28	0.31
Artificial	0.53	0.53	0.54
Combined	0.59	0.60	0.59
Generalized	0.22	0.13	0.12

Table 5.11 shows the confusion matrix of the predictions from generalized learning when using only the invariant features. We can see that the results are similar to the ones obtained when using the full feature set. However, we can see that the predicted classes are now slightly more varied, with more instances being predicted as ABC and ACO.

The poor results of the generalized learning model could be explained by the single-label setup of the machine learning experiment. Because a large number of the BBOB instances

Table 5.10: The list of 13 landscape features that were determined to be invariant to shifting and scaling, with the names taken from the flacco library.

cm_angle.angle.mean	ela_meta.lin_simple.adj_r2
ela_meta.lin_simple.intercept	ela_meta.lin_simple.coef.min
ela_meta.quad_w_interact.adj_r2	ela_meta.quad_simple.adj_r2
ela_meta.lin_w_interact.adj_r2	disp.ratio_mean_02
disp.ratio_median_25	nbc.nb_fitness.cor
pca.expl_var_PC1.cov_init	pca.expl_var.cov_init
pca.expl_var.cor_init	

Table 5.11: The confusion matrix for generalized learning, using only the landscape features that were found to be invariant to shifting and scaling. The columns represent true classes, while the rows represent the predictions.

Pred. \ True	ABC	ACO	CMA-ES	CSO	DE	FEP	GA	PSO	SA	Rand
ABC	0	0	0	0	0	0	10	0	0	0
ACO	0	2	6	7	0	3	6	0	0	0
CMA-ES	0	2	9	12	12	0	42	7	0	0
CSO	1	8	7	47	3	10	23	4	0	0
DE	1	0	3	6	2	0	2	4	0	0
FEP	0	0	4	3	3	0	2	7	0	0
GA	0	0	8	12	0	0	22	0	0	0
PSO	0	12	8	17	0	0	15	0	0	0
SA	0	0	1	3	0	0	0	0	0	0
Rand	0	0	0	1	0	0	3	0	0	0

have multiple algorithms for which there is no statistical difference in performance, it is possible that in some cases where the BBOB instances are miss-predicted, the algorithm that was predicted was not the absolute best performing, but instead one of the algorithms for which there was no statistical difference in performance.

However, the model achieves poor results even if the BBOB problems where multiple algorithms perform well are excluded from the test set and only the 140 BBOB instances with a single best-performing algorithm are used for evaluation. In this scenario, the accuracy achieved by a generalized learning model increases to 0.30 when using all features and 0.32 when using the features invariant to shifting and scaling.

To further account for this scenario, we conduct the second machine learning experiment where the models tested on the BBOB problems were reevaluated by considering all of the algorithms with no statistical difference in performance to the best algorithm as the correct prediction. This creates an experiment that is similar to multi-label classification. However, in our case, the trained model used is still the same as used in the original single-label experiments, and so only a single class can be predicted by the trained model. This separates our experiment from a true multi-label scenario.

Table 5.12 shows the results of this experiment for the three models that use the BBOB instances as the testing dataset.

Table 5.12: The results of the algorithm selection model when considering all algorithms without a statistically significant difference to the absolute best performing algorithm to be a correct prediction.

Model	Accuracy (All)	Accuracy (Invariant)
BBOB (Instance Split)	0.90	0.88
BBOB (Problem Split)	0.47	0.49
Generalized	0.40	0.43

Only accuracy is reported since per-class precision and recall metrics cannot be used with this experimental setup as there are multiple correct labels for each instance, i.e. multiple algorithms for which there is no statistical difference in performance when compared to the best-performing algorithm. We present both the variant of the experiment that uses all available landscape features, as well as the variant that uses only landscape features invariant to shifting and scaling. As only a single class is predicted, evaluation metrics used for multi-label classification such as Hamming loss also cannot be used. Because the artificial data set only contains a single correct label, these results do not include the scenario where the model is both trained and tested on the artificial instances. As the scenario where the model is trained and tested on a combined set of problems also includes the artificial instances, we also chose to exclude that scenario.

From the results, we can see that this increases prediction accuracy, as was expected. The BBOB instance split model now achieves an accuracy of 0.90, predicting most instances correctly. The BBOB problem split model now achieves an accuracy of 0.47 up from 0.21, and the generalized model achieves an accuracy of 0.40. These results are now close to the results achieved on the model that was both trained and tested on the artificial instances in the previous experiment. However, despite the improvement in the generalized model and the problem split model, we can see that both of them achieve an accuracy that is still much lower than the instance split model.

5.3 Complementarity Analysis

After the machine learning part of our methodology, we performed several additional experiments aimed at exploring the complementarity of the artificially generated problem set and the handmade BBOB problem set in order to better explain our machine learning results. We split these experiments into two groups. In the first group, we examine the BBOB and the artificially generated problems by performing correlation analysis as described in (Eftimov, Popovski, et al., 2020). This allows us to directly examine the correlations between the individual instances. Ideally, instances that are solved well by the same algorithm should be correlated with each other. In the second group, we examine the algorithm selection models using Shapley Additive Explanations. This framework for interpreting model predictions assigns to every landscape feature of a particular instance a value that describes how much this specific landscape feature contributed to the decision of the model to classify the instance as a particular class. This allows us to compare which landscape features the different models consider to be important, as well as to determine which landscape features most contribute to the classification of every individual class.

5.3.1 Correlation Analysis

To present the results of the correlation analysis, we plot the correlations between different problems. If the problem sets are distinct, we would expect the problems within one problem set to be correlated with one another, but that there should only be a small amount of correlation between the problems from a different problem set. As our dataset contains a relatively large amount of data, visualizing all instances would not provide readable visualizations. Instead, we only use a subset of all instances for the analysis in this section. Specifically, we randomly sample the artificial and BBOB instances, while preserving the class distribution of the whole set. For the artificial instances, we sample two instances from every class, for a total of 20 instances. For the BBOB instances, we sample one instance from each base problem, for a total of 24 instances. This results in 44 sample instances that will be used for the visualization. Because the procedure of sampling the instances is stochastic, we repeat this procedure three times to obtain three alternative samplings. We use these samplings to show multiple examples of the resulting visualizations.

The results of the correlation analysis calculated using Pearson correlation are shown in Figures 5.2, 5.3, and 5.4. The figures show the correlation between the BBOB instances and the artificial instances, using three different sets of 44 instance samples that were generated as described earlier in this section.

While the three figures are not exactly the same, we can see that their structure is broadly the same. We can see that there is a large correlation between most of the artificial problems, with some exceptions, for example the first ACO and PSO instances in Figure 5.2. However, the classes these exceptions belong to vary based on the problems that are sampled.

The BBOB instances are less sensitive to sampling, as the sampling of the BBOB instances is performed based on the base BBOB problem. Because of this, all three visualizations produce similar results for the BBOB instances. We can see that the BBOB instances form two distinct groups where instances are correlated with each other, but not with the instances in the other group.

The correlation between the artificial and the BBOB instances produces interesting results. We can see that of the two BBOB groups, one is correlated with the majority of the artificial problems, while the second is correlated with the minority of artificial problems. In general, instances with a specific best-performing algorithm are not correlated with one

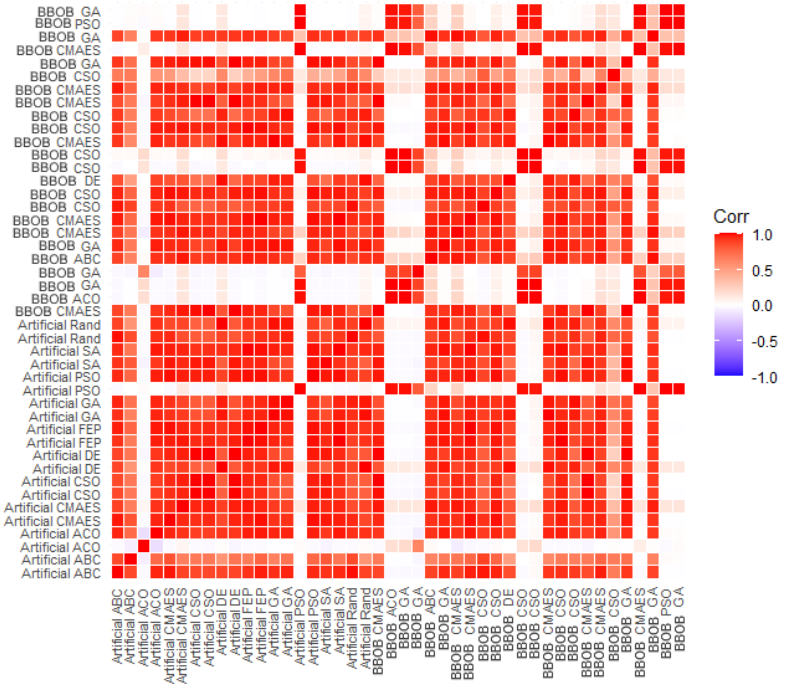

Figure 5.2: Correlation between the artificial and the BBOB problems, after the BBOB problems have been projected to the SVD subspace of the generated problems, using all landscape features calculated on the first sampling set. The labels describe the problem set that the features belong to, as well as the class of the instance (the algorithm that found the best final solution on the instance).

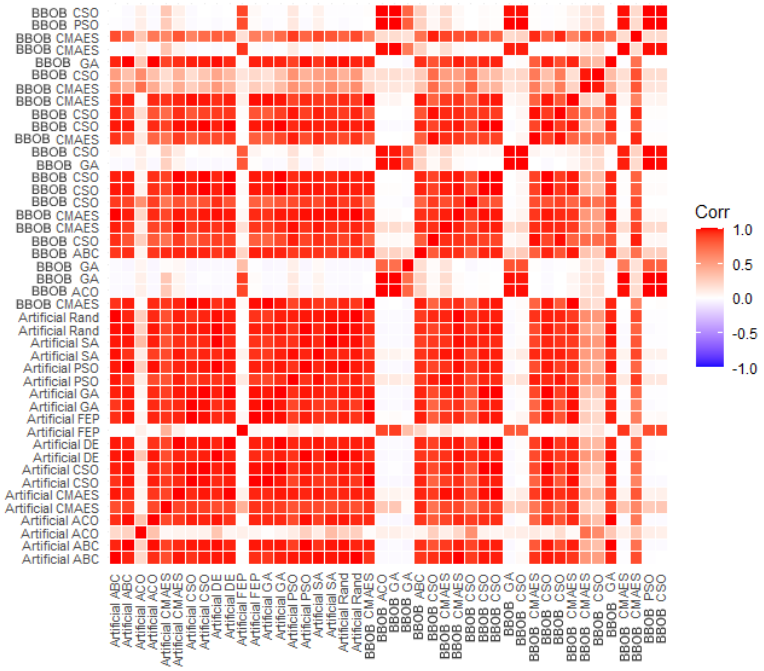

Figure 5.3: Correlation between the artificial and the BBOB problems, after the BBOB problems have been projected to the SVD subspace of the generated problems, using all landscape features calculated on the second sampling set. The labels describe the problem set that the features belong to, as well as the class of the instance.

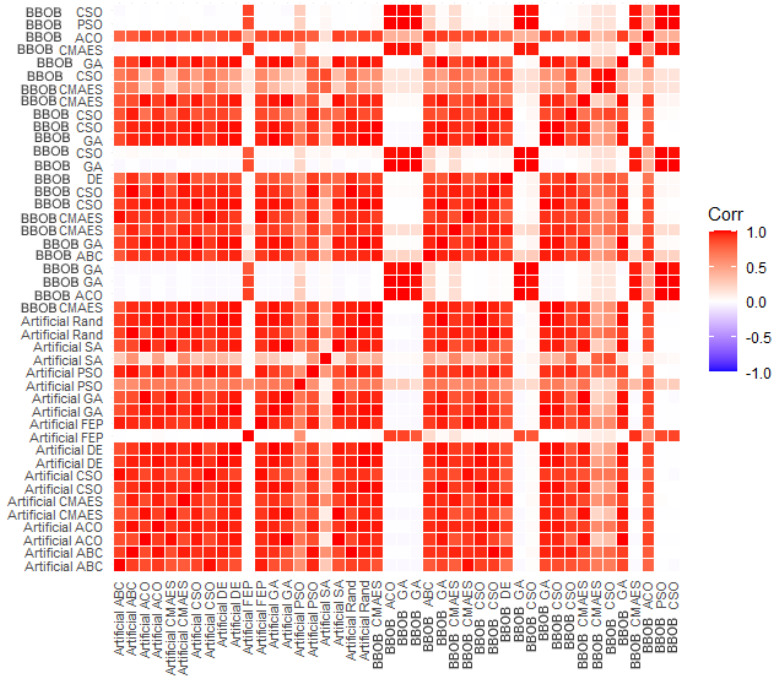

Figure 5.4: Correlation between the artificial and the BBOB problems, after the BBOB problems have been projected to the SVD subspace of the generated problems, using all landscape features calculated on the third sampling set. The labels describe the problem set that the features belong to, as well as the class of the instance.

another, which explains the poor results of the generalized model.

5.3.2 Analysis Using Shapley Additive Explanations

When analyzing the SHAP values, we are particularly interested in comparing the SHAP values of two different models: the generalized model and the instance split BBOB model. Specifically, we want to compare a model that performed well (the instance split model), with a model that achieved poor performance (the generalized model) in order to determine why the generalized model performed so poorly.

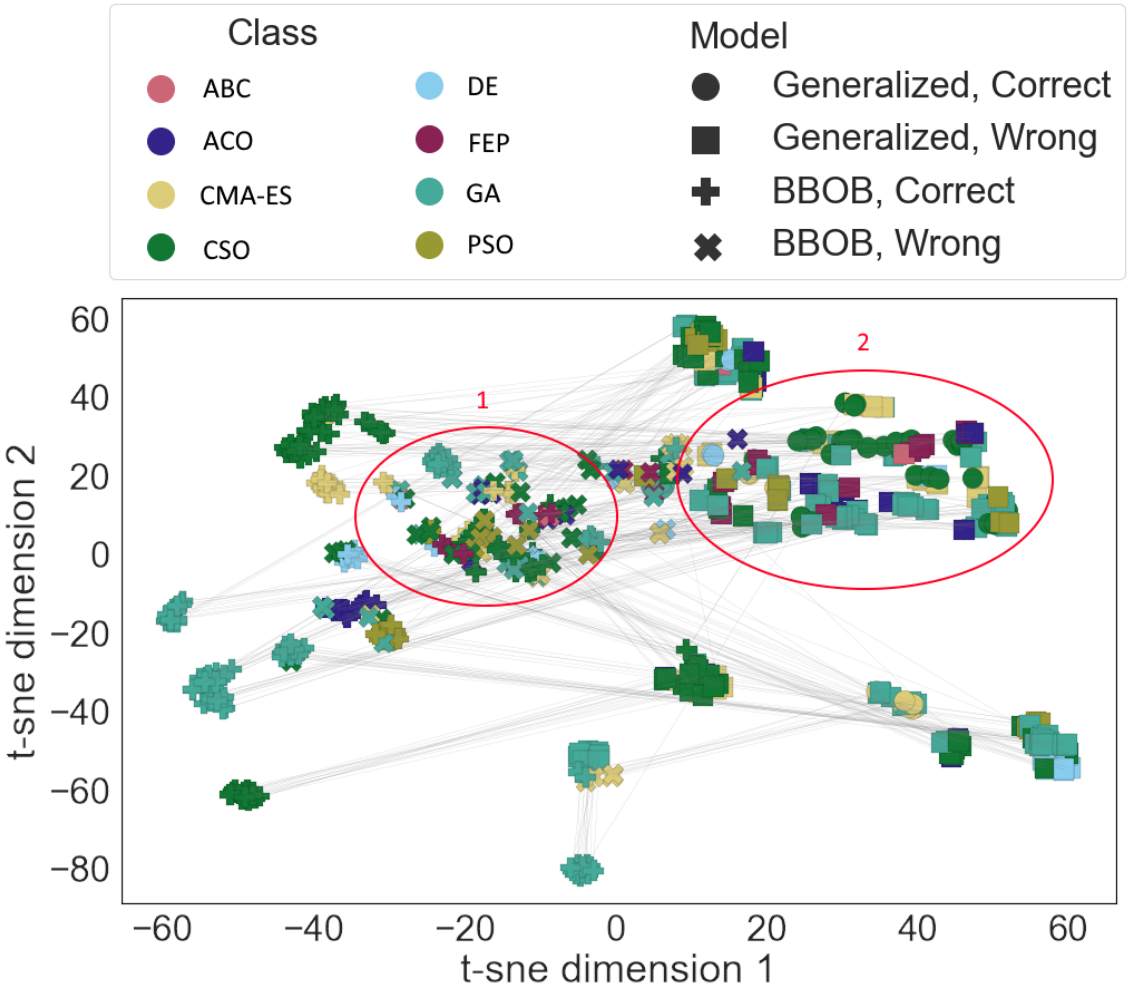

Figure 5.5: A t-sne representation of all predictions by both the instance split BBOB model and the generalized model, with each point representing a single instance. The shapes of the points represent the model used, and whether or not the prediction of the model was correct. The color represents the correct class of each instance. We highlight two groups of instances that we will examine in more detail.

Figure 5.5 shows the t-sne visualization of the SHAP values of the two machine learning models based on the predictions on the BBOB instances. The two models examined are the instance split BBOB model and the generalized model. Each point in the figure represents a single prediction of the BBOB instances. The color of each point shows the correct class for that instance, and the shape of the point represents both which of the two models made the prediction, as well as whether or not the prediction was correct. From the figure,

we can see that the SHAP values of the BBOB model mostly split the prediction into well-defined groups, with the groups corresponding to the correct class. However, there is a group in the middle of the figure, marked 1, which does not correspond to a single class, and which accounts for the majority of the incorrect predictions.

The generalized model shows a similar structure, with a large group in the middle right side of the figure, marked 2, and smaller clusters spread throughout the figure. However, unlike the BBOB model, the generalized model achieves a much worse accuracy, with the smaller groups not corresponding to the correct class.

Finally, we also note that the two models have almost no overlap with one another, which means that the generalized model prioritizes different features than the BBOB model when making predictions, which helps explain the poor generalized model performance.

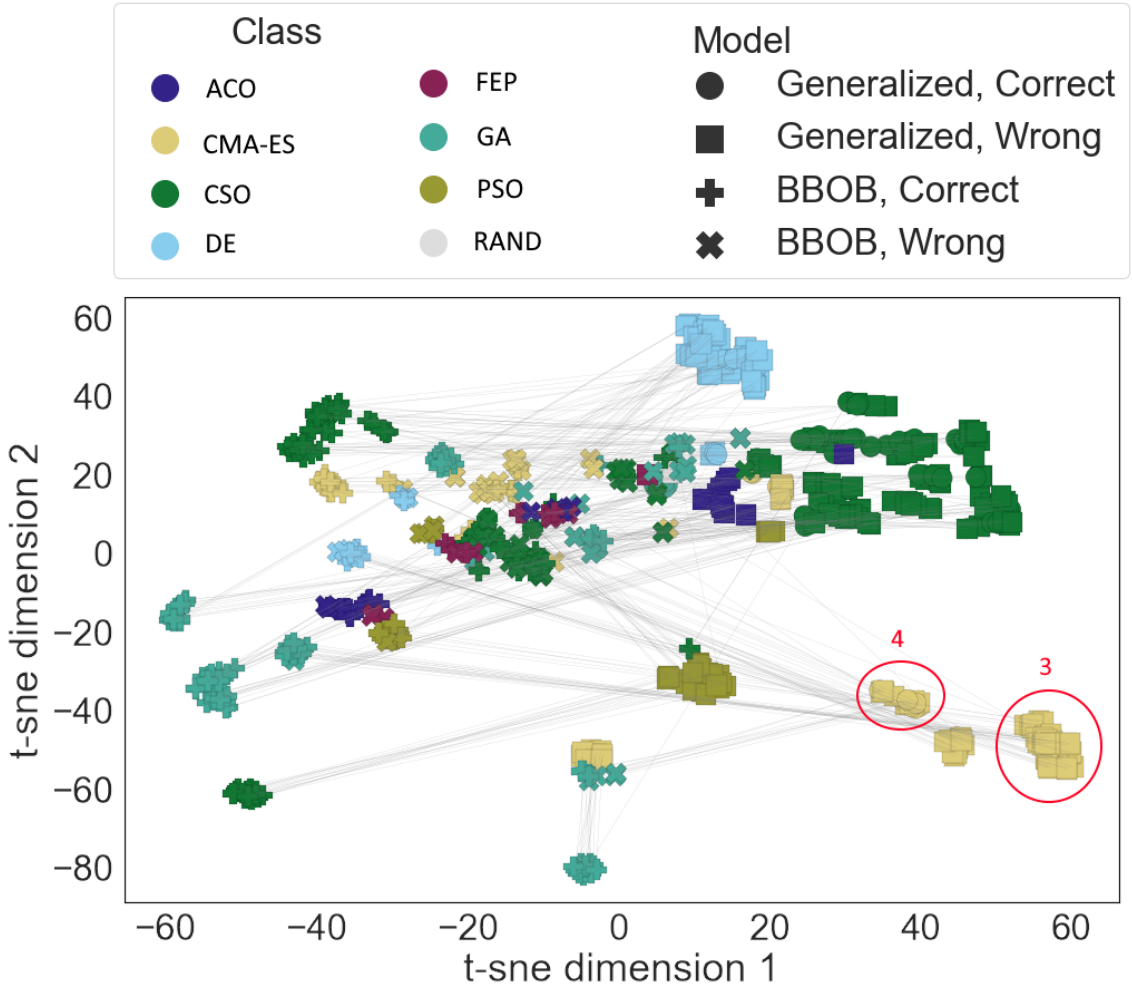

Figure 5.6: A t-sne representation of all predictions by both the instance split BBOB model and the generalized model, with each point representing a single instance that was predicted by the models. The shapes of the points represent the model used, and whether or not the prediction was correct. The color represents the predicted class of each instance. We highlight two groups of instances that we will examine in more detail.

Figure 5.6 shows the same visualization, but with the points colored by the predictions of both of the models. We can see that the BBOB model predicts most of its individual clusters correctly. However, it struggles to predict the large, central cluster, predicting most of its instances as CSO. We can see now why the generalized model performs poorly.

The large group on the top right is predicted entirely as CSO, while the smaller group at the top is predicted entirely as DE, and the group on the bottom right as CMA-ES. In reality, all of these clusters represent multiple different classes. It appears that the generalized model has given more importance to landscape features which might not correlate with algorithm performance.

In this Figure, we would also like to draw attention to two additional groups of the generalized model, marked 3 and 4. These groups appear interesting because a single group of the generalized model corresponds to multiple groups of the BBOB model. Because these relations are difficult to see in this Figure, we examine them in more detail.

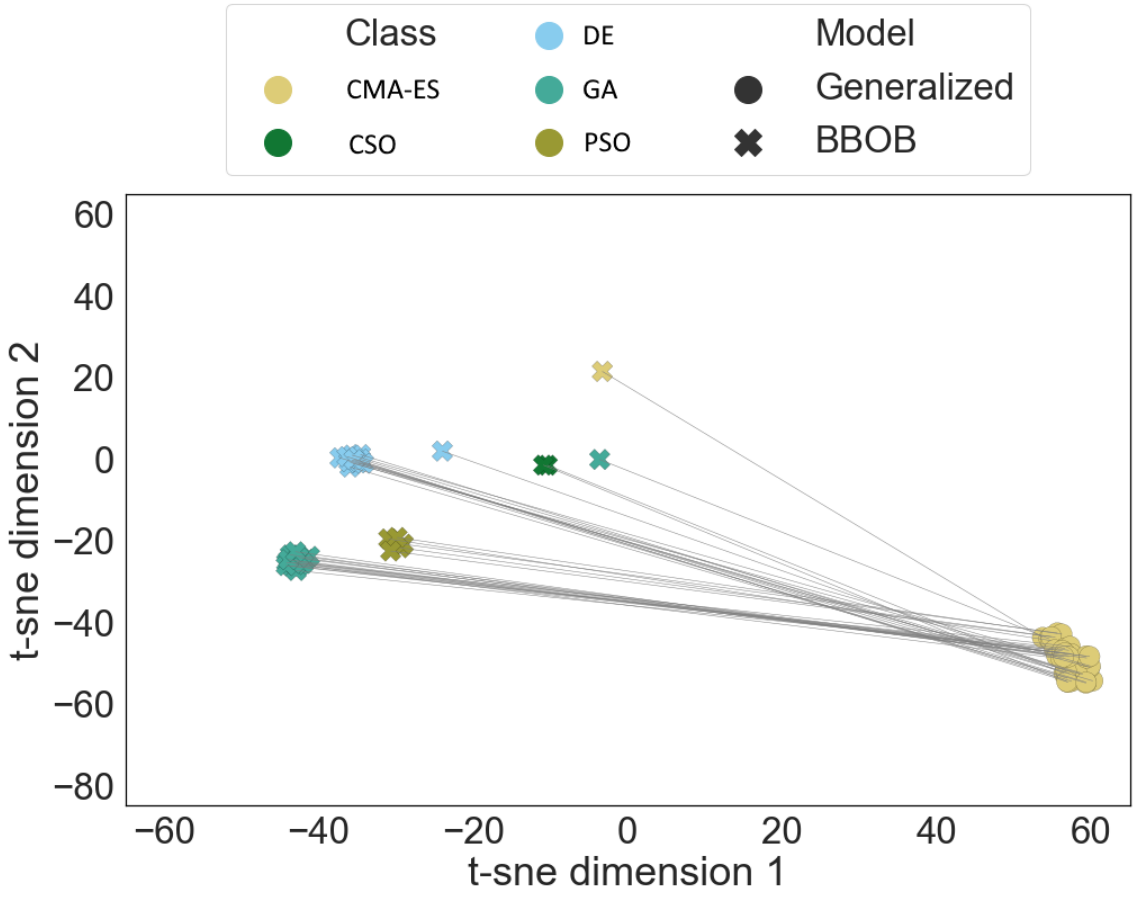

Figure 5.7: A smaller subset of SHAP values showing just the first bottom right group of the generalized model, as well as the paired BBOB model predictions. Colors represent the predicted class.

In Figure 5.7, we examine the group which was marked with 3 in Figure 5.6. We can see that the predictions of the generalized model from this group belong to roughly three different groups from the BBOB model. The three separate groups of the BBOB model are able to correctly predict the best-performing algorithm. On the other hand, the single group belonging to the generalized model mispredicts most of the instances. This indicates that the generalized model is unable to distinguish between these problems, while the BBOB model can. We can also see that there is a single prediction of the BBOB model that is predicted as CMA-ES, while the generalized model incorrectly predicts the entire group as CMA-ES.

Figure 5.8 shows the small group located immediately to the left of the group we

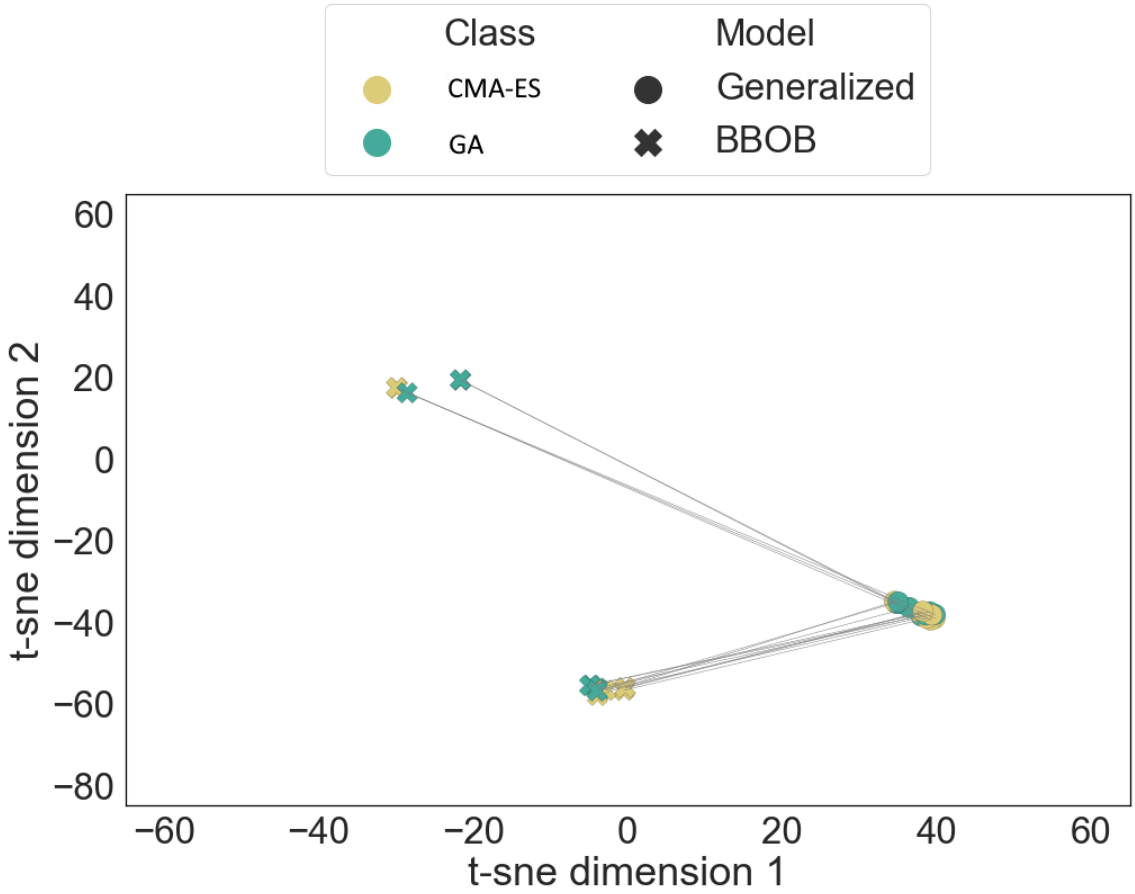

Figure 5.8: A smaller subset of SHAP values showing just the second bottom right group of the generalized model, as well as the paired BBOB model predictions. Colors represent the predicted class.

examined in Figure 5.7, which was marked with 4 in Figure 5.6. Here, we once again see that the group belonging to the generalized model maps to multiple BBOB groups. However, in this case, both the BBOB and the generalized groups contain predictions mixed between classes 3 and 7.

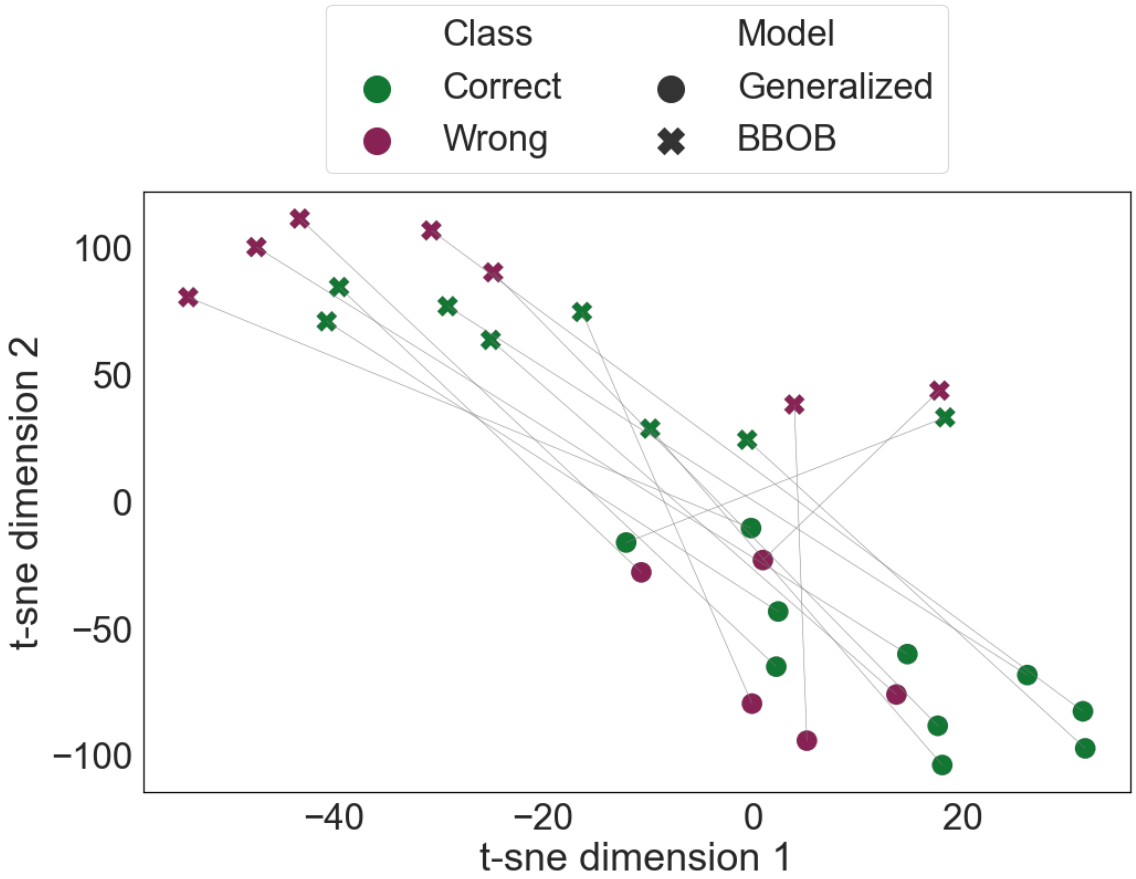

Figure 5.9: The SHAP values from Figure 5.8, but visualized further apart, and colored by whether or not the model predictions were correct.

Figure 5.9 shows an alternative view of Figure 5.8, with individual points plotted further apart, and colored by whether the prediction of the machine learning model was correct or incorrect. When comparing the pairs of instances from the generalized and the BBOB model, we can tell that there are a number of instances that the generalized model predicted correctly, but the BBOB model did not. This shows that even the generalized model is able to correctly predict some instances that the BBOB model could not.

Figure 5.10 shows the same visualization as the previous figures, but with points simply colored by whether or not a prediction was correct. Figure 5.11 show the same situation, but using the multi-label scenario instead. We can observe that switching to a multi-label scenario improves the results in the two central groups that were previously classified mostly incorrectly. From this, we conclude that the instances belonging to these groups are those where multiple algorithms achieve very similar performance. We also see some improvements in other groups of the artificial model, but not to the extent that is present in the central group.

Figures 5.12 and 5.13 show two different comparisons involving the problem split BBOB model rather than the instance split model that we have examined so far.

Figure 5.12 compares the problem split model with the generalized model. We can see

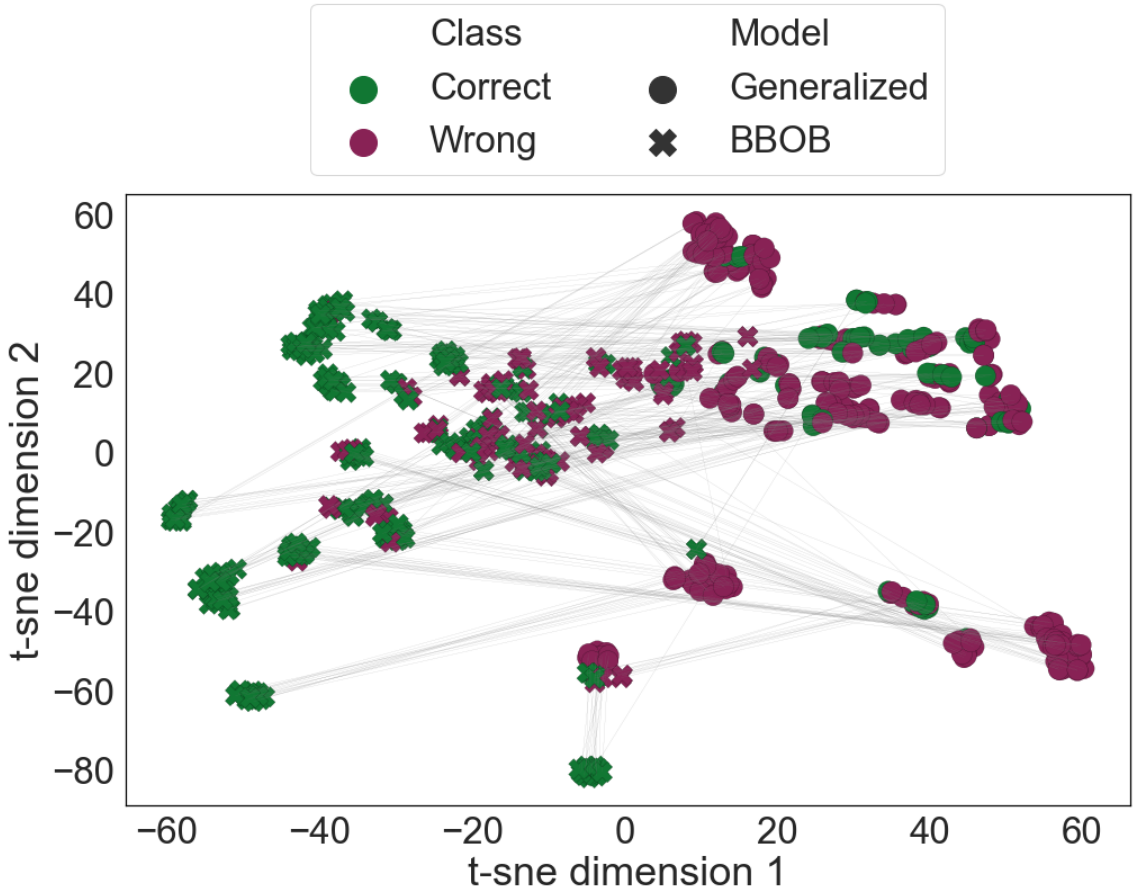

Figure 5.10: A t-sne representation of all predictions by both the instance split BBOB model and the generalized model, colored by whether or not the predictions were correct.

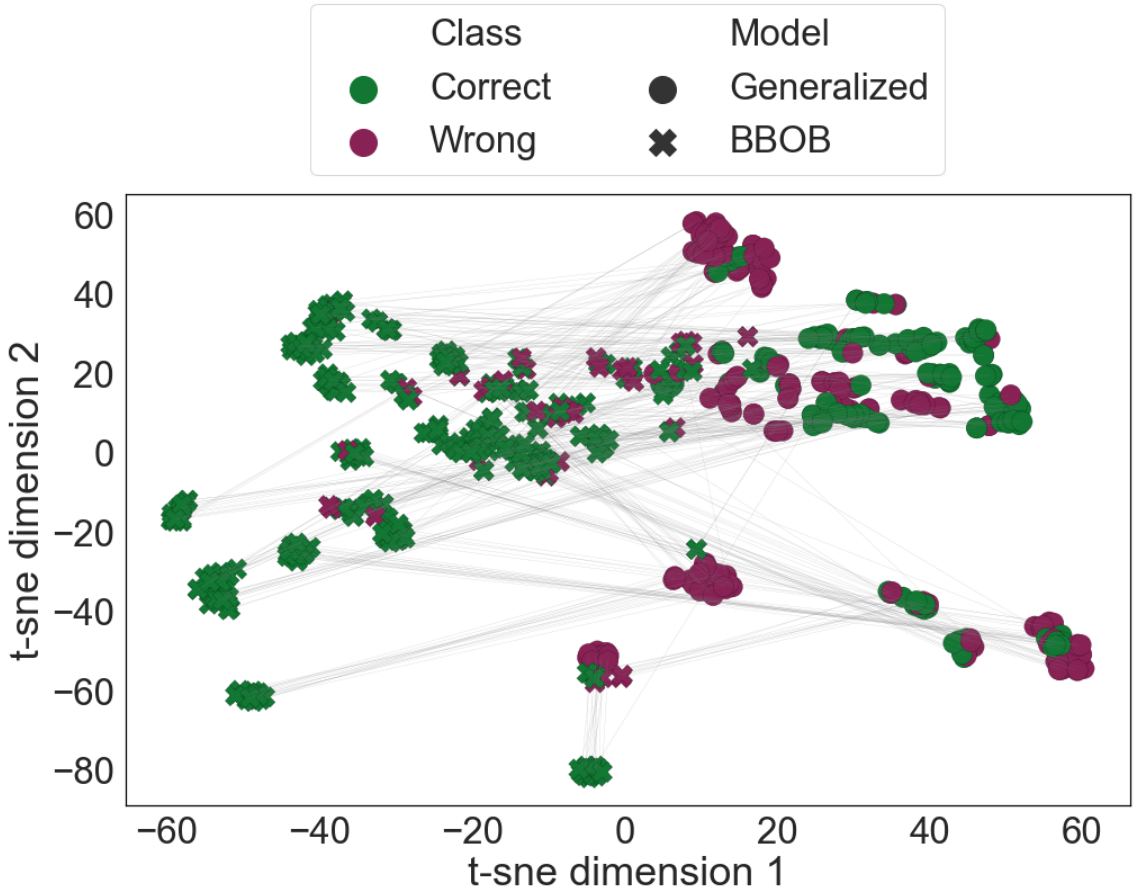

Figure 5.11: A t-sne representation of all predictions by both the instance split BBOB model and the generalized model, colored by whether or not the predictions were correct using the multi-label evaluation.

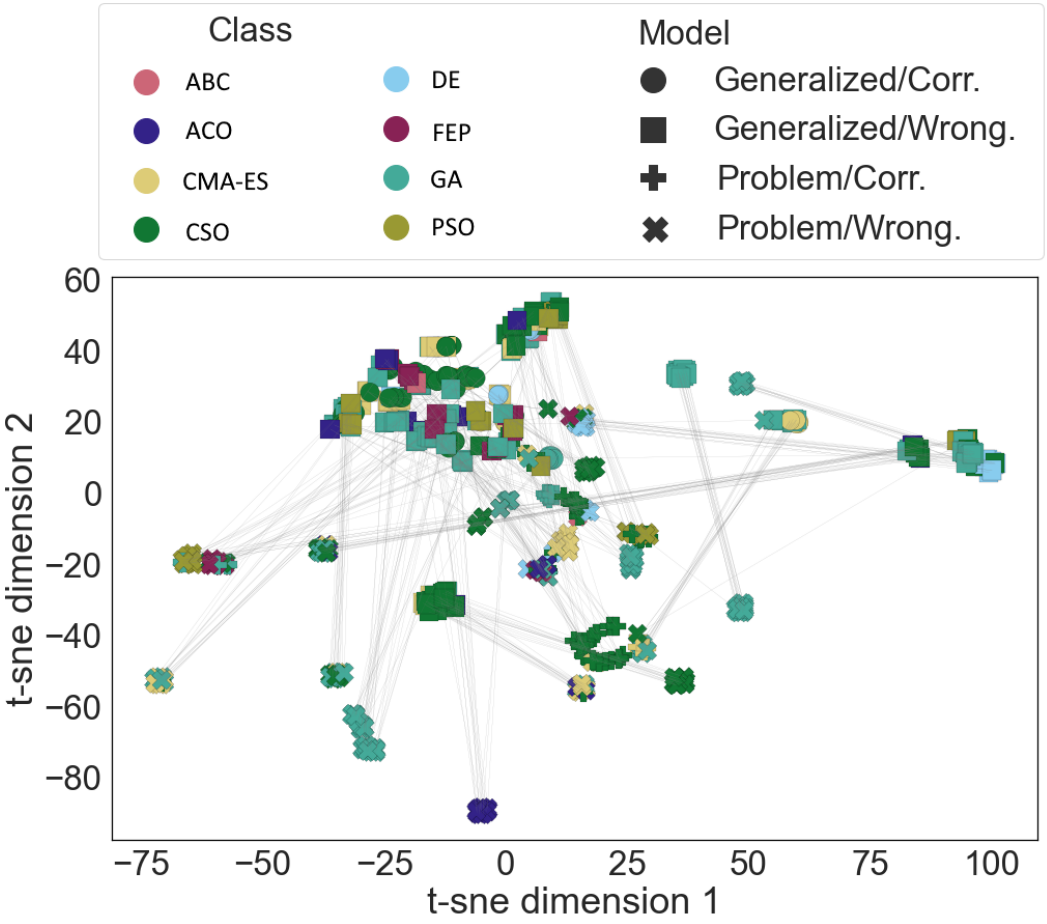

Figure 5.12: A t-sne visualization of SHAP values showing the comparison between the generalized model and the problem split BBOB model.

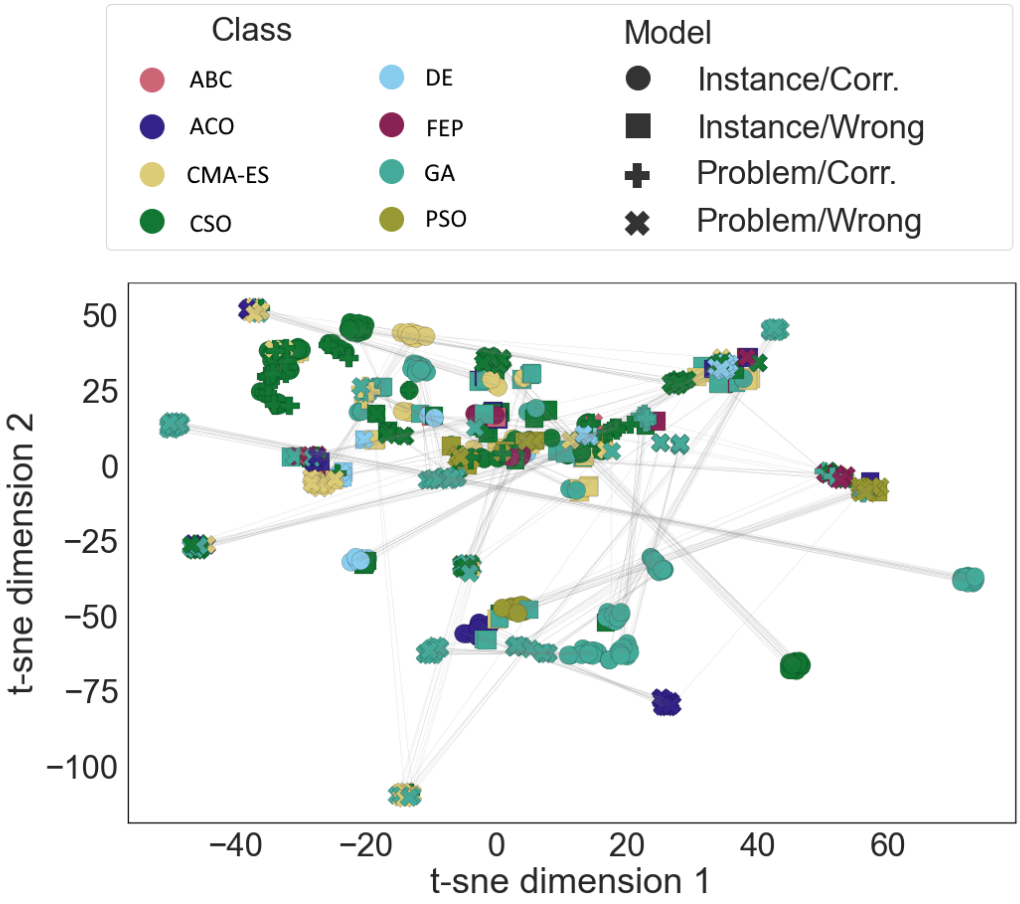

Figure 5.13: A t-sne visualization of SHAP values showing the comparison between the instance split BBOB model and the problem split BBOB model.

that despite the fact that both models achieved similarly poor performance, the SHAP values that the two models produce are still fairly different. As we have seen in previous comparisons, both of these models have a large group of incorrect predictions, as well as multiple smaller clusters. However, the smaller clusters are distinct from one another, and there is no large overlap between the two models.

Figure 5.13 compares the problem split BBOB model with the instance split BBOB model. Here, we can see that the models do share some similarities. The large group of incorrect predictions is located in a similar space for both models. This indicates that neither of the models was able to correctly differentiate between the problems in this cluster. To the left of this cluster, we see another shared cluster belonging to the class CSO which was predicted correctly by both models.

This can be explained by the fact that the instance in this cluster belongs to two different base problems. As such, the problem split model was still able to learn their characteristics even with one base problem missing from its training set, because the other base problem that forms this cluster will still be included in the training set. However, this is the only cluster shared by both models. This lack of overlap between the models was expected due to the poor performance of the problem split model.

Figures 5.14, 5.15, and 5.16 show the mean SHAP values of correct predictions of the generalized and the instance split BBOB models, separated by the class of the prediction. Figure 5.14 shows the mean values for instances belonging to the class CMA-ES, Figure 5.15 for the class CSO, and Figure 5.16 for the class DE.

We can see that there are large differences between the generalized and the BBOB models, which is unsurprising given the poor performance of the generalized model. For the CMA-ES class, the landscape feature *ic.h.max* is the most important feature for the BBOB model, while it is not considered important by the generalized model. The feature *ic.eps.max* is likewise important for the BBOB model, but on average gives negative contributions to the generalized model. A similar result can be observed for the features *limo.avg_length.reg* and *limo.length.mean*. On the other hand, the generalized model considers the features *disp.ratio_median_25*, *disp.diff_median_25*, and *disp.diff_mean_25* as much more important.

For the CSO class, we see that the generalized model considers the feature *cm_angle.dist_ctr2best.mean* to be by far the most important, while the BBOB model almost does not consider it at all. This is particularly notable because, as we have seen in the confusion matrix in Table 5.8, the CSO class accounted for a large part of all of the predictions. This could indicate that a large part of the correct predictions of the generalized model was based largely on a single feature. We can verify this by looking at the mean SHAP values for all correct predictions of the generalized model, which are shown in Figure 5.17, where we can see that this feature is a large outlier in terms of feature importance. For the BBOB model, *ic.eps.max* is again considered as one of the most important features, but it is overtaken by *nbc.nn_nb.cor*. Generally speaking, the BBOB feature importance of this class looks roughly similar to the CMA-ES class.

For the DE algorithm, we once again see that the features *ic.eps.max*, *limo.avg_length.reg*, and *limo.length.mean* are considered important for the BBOB model while having negative values for the generalized model. We can also see the feature *cm_angle.dist_ctr2best.mean* being more important for the generalized model, although not to the same extent as observed for the CSO class. However, here we see much larger differences in the feature importance of the BBOB model. In particular, *ela_meta.lin_w_interact.adj_r2* is now the most important feature. In the generalized model, the most important feature is *ela_meta.lin_simple.coef.max_by_mean*, which was considered unimportant by both the generalized and the BBOB models in the previous two classes.

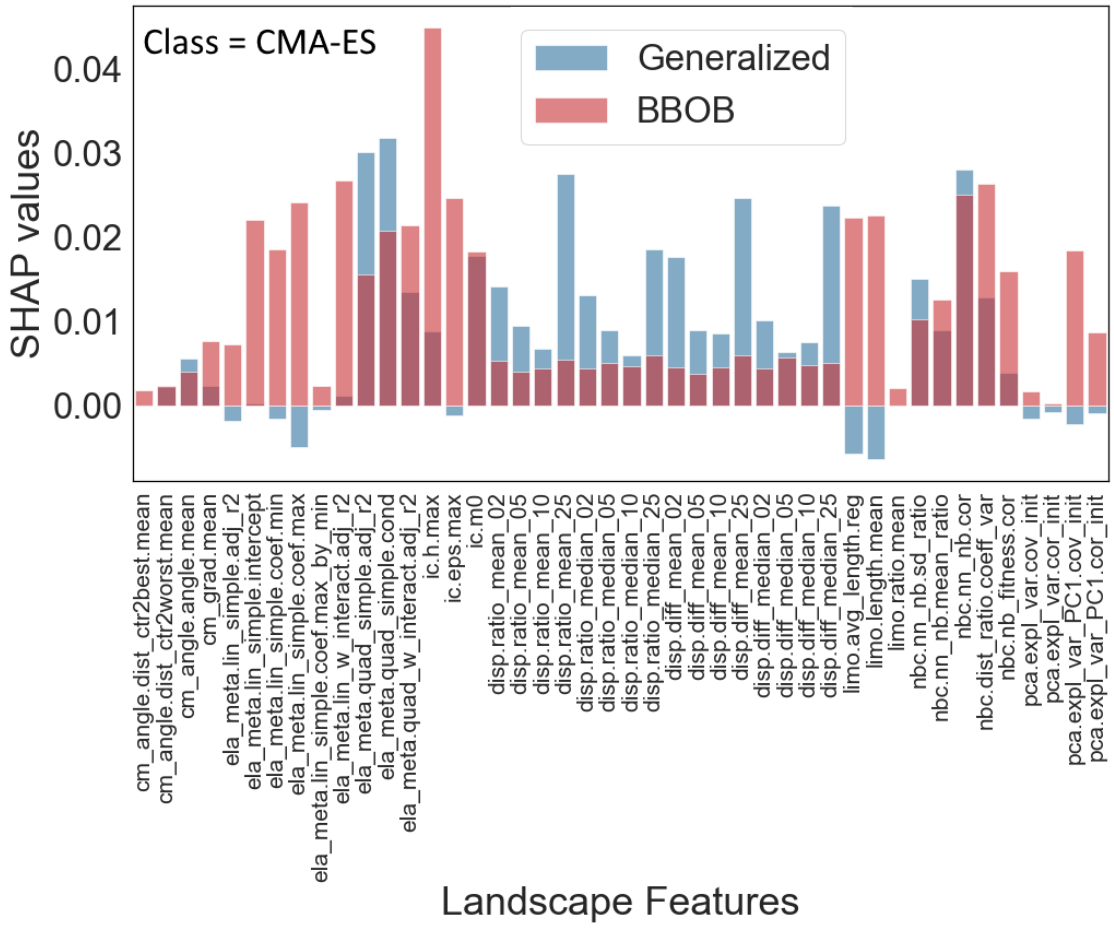

Figure 5.14: Mean SHAP values of the generalized and the BBOB instance split models on the correct predictions of instances with the class CMA-ES.

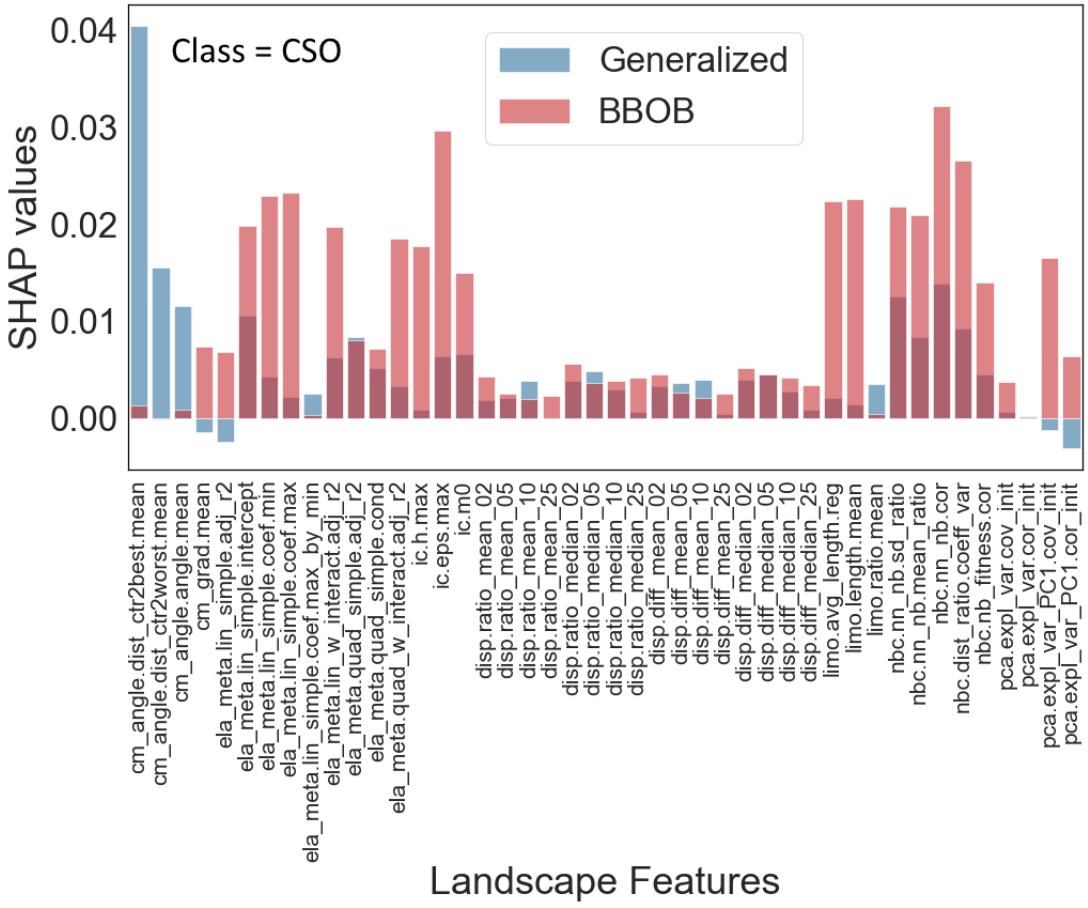

Figure 5.15: Mean SHAP values of the generalized and the BBOB instance split models on the correct predictions of instances with the class CSO.

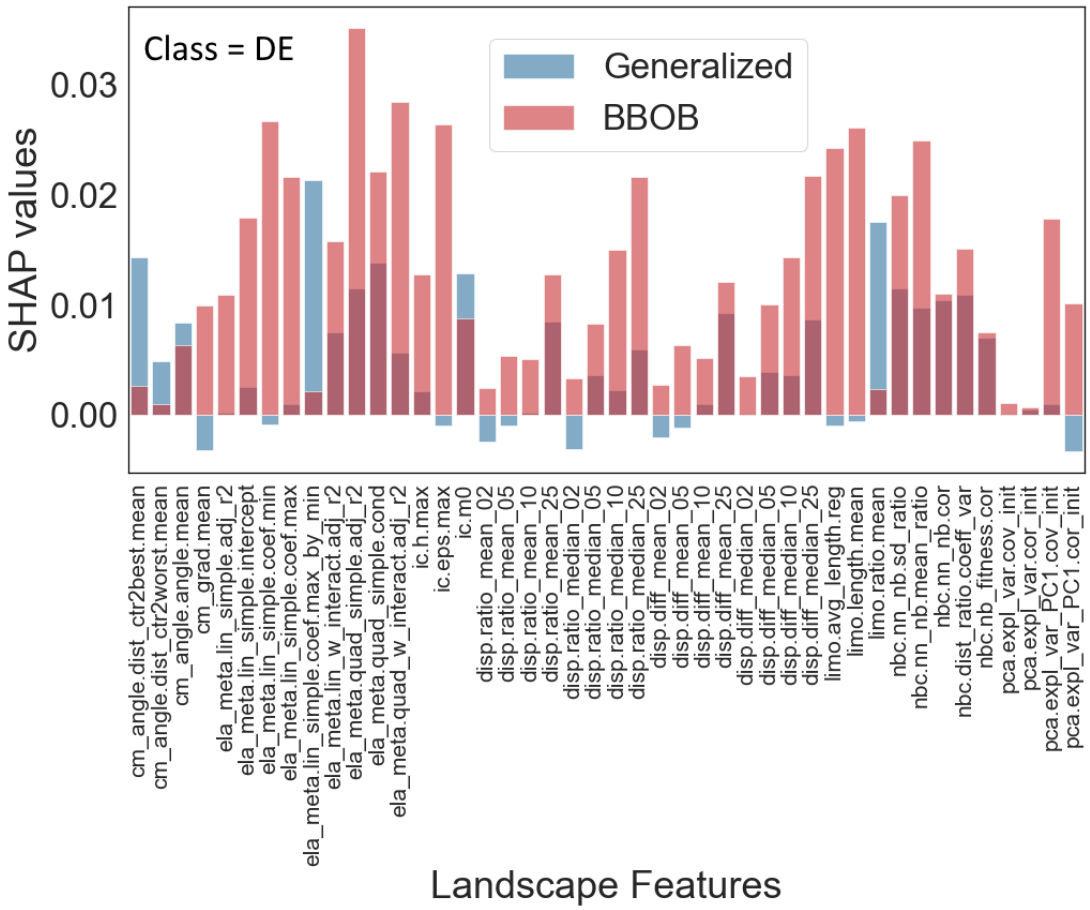

Figure 5.16: Mean SHAP values of the generalized and the BBOB instance split models on the correct predictions of instances with the class DE.

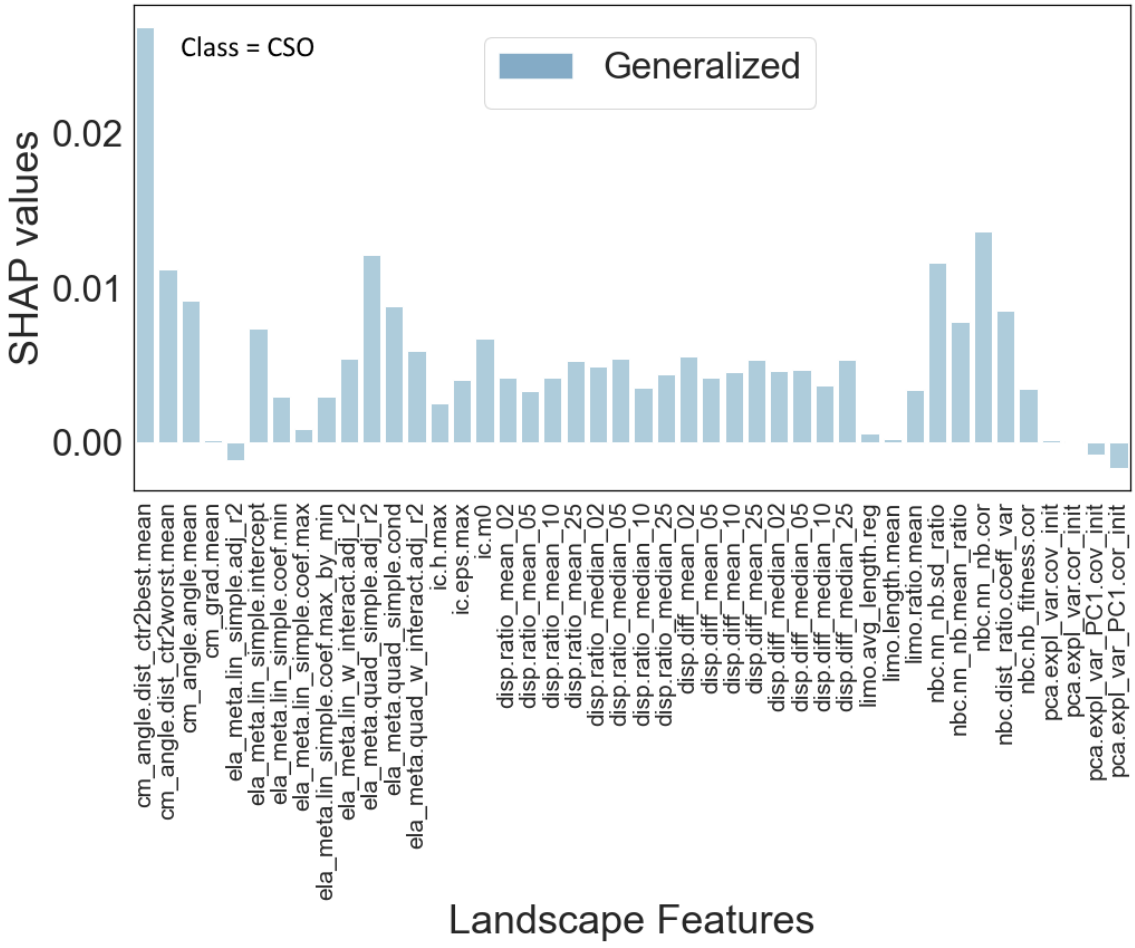

Figure 5.17: The mean SHAP values of all correct predictions of the generalized model.

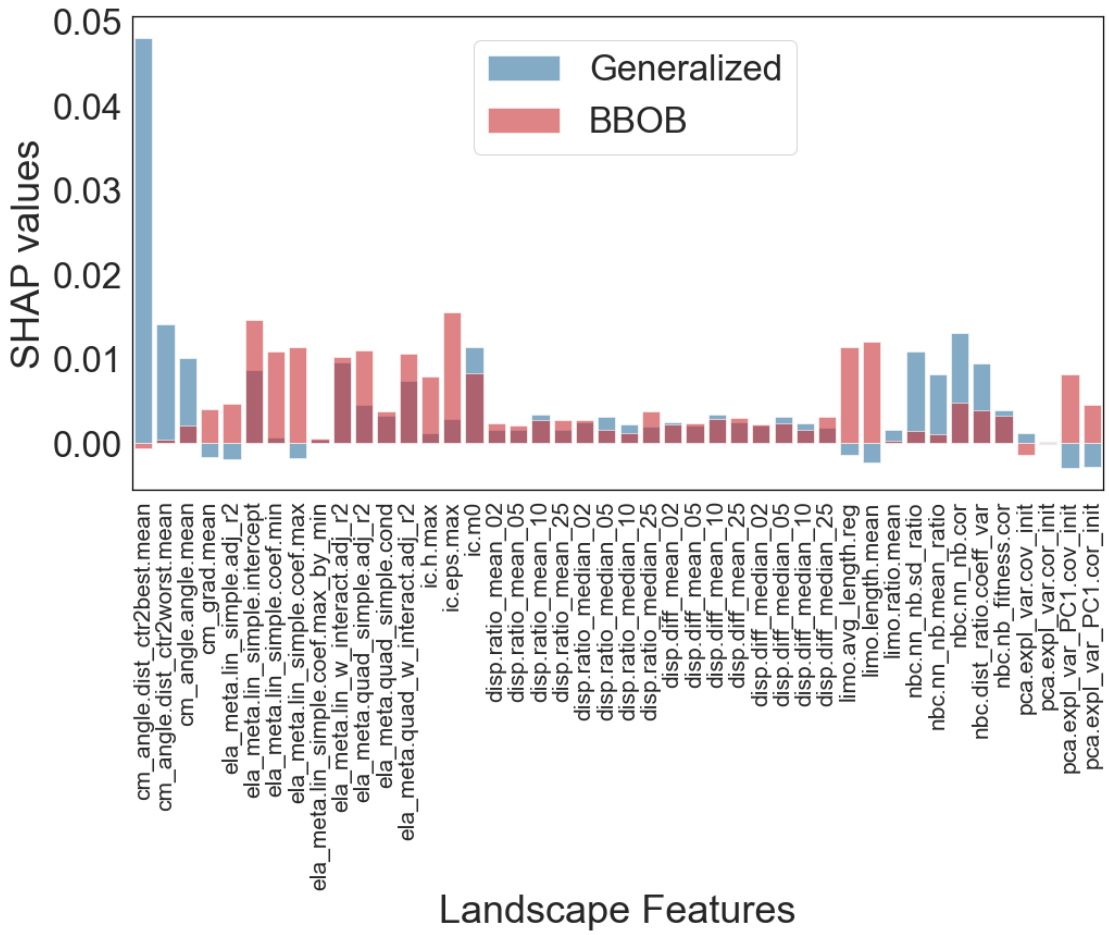

Figure 5.18: Mean SHAP values of the generalized and the BBOB instance split models on the false positive predictions of instances with the class CSO.

As the CSO class represents the majority of the predictions of the generalized model, we examine it more deeply in Figure 5.18. This figure shows the SHAP values of all predictions of the CSO class which were false positives, rather than the correct predictions that we examined in the previous figures. The BBOB model shows low feature importance for all features, which indicates that when the BBOB model makes false positive predictions, there is no large support for any particular class. But in the generalized model, we now see an even larger outlier in the feature *cm_ angle.dist_ ctr2best.mean*. It is likely that the overreliance of the model on a single feature is one of the factors which contributed to its poor performance.

5.4 Discussion and Conclusion

In this chapter, we have analyzed the performance of automated algorithm selection using a variety of different models. We have shown that while a model trained and tested on the same dataset performs well, models that are trained on a set different from the one they are tested on perform poorly. We then performed a more thorough evaluation of the performance of these models, specifically by using SHAP to examine their feature importance. We found that the different models have large differences in the features they consider important.

In the first part of this chapter, we have combined the knowledge of the performance and the problem space of optimization by performing algorithm selection using ELA. Specifically, we examined how algorithm selection models perform when trained and tested on two different problem sets: an artificially generated set, and a handmade set of BBOB problems.

We first evaluated several models that were trained and tested on the same problem set. When the models were trained and tested on the artificial problem set, the model achieved good results. When the models were trained and tested on the BBOB problem set, the performance of the models varied greatly based on the way cross-validation was performed. If the cross-validated training set was created so that it always contained instances from every BBOB base problem, the algorithm selection results were very good. But when the cross-validated training set was created by excluding all instances of a single BBOB base problem, the results were very poor.

This shows that the choice of the training and testing data, even when using a single problem set, is hugely important to algorithm selection results and that the problems in the BBOB set are so different from one another that it is generally not possible to generalize the information learned from 23 of the BBOB problems to the missing 24^{th} problem.

Then, we evaluated the generalized algorithm selection model that was trained on one set of problems (the artificial problems) and tested on a completely different set (the BBOB problems). This model performed poorly, likely due to the differences in the two problem sets. The poor results of this generalized algorithm selection model reinforce the findings of the previous chapter, which showed that landscape features have problems generalizing between different problem sets, at least when using certain problem sets that we have examined in this thesis.

Specifically, we have shown that an algorithm selection model trained and tested on artificial problems achieved an accuracy of 0.54, but that evaluating the same model on the BBOB problem set achieved a much lower accuracy of 0.20. This shows that the information learned from the artificial problem set using landscape features cannot be generalized to the BBOB problem set. We have additionally performed the same experiment by using only the landscape features that were found to be invariant to the transformations of shifting and scaling in the previous chapter. Doing so produced a slightly higher accuracy.

While the improvement is small, the fact that the performance does not degrade when using the invariant features shows that these features themselves carry all of the necessary information for predicting algorithm performance.

In the second part of this chapter, we conducted additional experiments in order to further explain the algorithm selection results. We did this first by using correlation analysis to examine the correlations between the problems from the two different problem sets. We then used SHAP features to compare the different machine learning models based on their feature importance. By analyzing the SHAP values of the generalized model and the BBOB instance split model, we have shown that the two models consider different landscape features when making their predictions, which helps explain the poor results of the generalized model.

Additionally, we have shown that the generalized model is not capable of distinguishing between different groups of instances in the same way that the BBOB model can, which leads to mispredictions of the problems belonging to these groups. Interestingly, the generalized model is able to correctly predict some problems that the BBOB model cannot. When examining the mean SHAP values of the correct model predictions, we have likewise found large differences between feature importance of the generalized and the BBOB model. In particular, we found the feature cm_angle.dist_ctr2best.mean to be a huge outlier in terms of feature importance for the generalized model.

It is possible that the generalizability of our algorithm selection experiments could be improved by a better choice of artificial problem generator. While we have attempted to create a balanced set of artificial problems by ensuring a balanced distribution of classes, this only created a balanced distribution according to the performance space of the algorithms, and not the problem space. However, this, as well as the requirement that generated problems should only have a single best-performing algorithm, likely introduced an additional bias to our set of problems. Because of this, it is possible that a problem generator that could create a set of problems balanced in terms of their landscape features would produce more generalizable results. It would also be interesting to attempt to perform similar generalizability experiments on real-world problems rather than the BBOB benchmark problems. Unfortunately, real-life problems are hard to acquire in large numbers.

Chapter 6

Conclusions and Future Work

It has long been known that the performance of optimization algorithms is heavily dependent on the specific problems that the algorithms are solving and that no single algorithm can exist that can solve every algorithm perfectly. This means that the task of choosing the correct algorithm for a specific task, called algorithm selection, is a crucial part of obtaining good optimization results. In this thesis, we have advanced the knowledge of two areas required to achieve good algorithm performance. These are the knowledge of optimization algorithms (the performance space of optimization), of the problems that the algorithms would be solving (the problem space of optimization), and of how to combine the knowledge of both the performance and the problem space with algorithm selection. We have analyzed both of these spaces and presented the drawbacks and challenges present in current state of the art.

This chapter serves as a final overview of the work presented in this thesis. We first give a brief summary of all of the contributions of this thesis. Then, we evaluate the hypotheses that we defined in the introduction of this thesis. Finally, we discuss our plans for future work.

6.1 Summary of the Contributions

In the first part of this thesis, we examined the performance space of optimization by evaluating two of the most popular numerical single-objective optimization benchmarks. These optimization benchmarks are commonly used to evaluate newly developed optimization algorithms, with yearly benchmarking events allowing comparisons with other recently developed algorithms.

By analyzing the performance of algorithms from these two benchmarks, we have shown that there has been no statistically significant improvement in overall algorithm performance from 2013 to 2018 and that the development of new algorithms has mostly been incremental rather than presenting entirely new ideas. These results also reinforce the importance of algorithm selection. Further, they show that one issue with optimization benchmarks that employ only a limited set of optimization problems is that certain algorithms might appear to perform better overall than they do in practice because they have been designed with the specific benchmark problems in mind.

We have also shown that depending on the problem set, some older algorithms can achieve performance that is better than more recent algorithms. This calls into question the common practice of comparing newly developed optimization algorithms only with the most recent state of the art, for example comparing algorithm performance only with the best performing algorithms from the single previous year of some benchmark competition.

In the second part of this thesis, we shifted our attention to the analysis of the optimization problem space. A technique called Exploratory Landscape Analysis has recently received a large amount of attention as it allows us to transform problem samples into numerical descriptors of an optimization problem that are called landscape features. This technique has shown great results in specific use cases, such as in predicting the performance of algorithms on a specific set of benchmark problems. However, some prior work has shown that ELA might have problems generalizing between different problem sets.

We have expanded this analysis by showing that certain landscape features are affected by the transformations of shifting and scaling. These two transformations are relatively simple, and we believe they should not have a large effect on most state-of-the-art optimization algorithms. This means that the information provided by the landscape features that are affected by these transformations might not correlate with algorithm performance. Additionally, we extended this analysis by examining how the invariance of shifting and scaling is affected by other factors, specifically the choice of sampling size, sampling strategy, dimension, and the problem set used. In doing so, we found the sampling strategy to have the largest effect, while the other factors had only a small or no effect.

We have verified these findings by conducting a complementarity analysis of the problem sets used in the two benchmarks examined in the first part of the thesis, by visualizing two different problem sets on a 2D plane using t-sne and principal component analysis. Using all available landscape features produced a poor visualization, and the features that were affected by shifting and scaling had to be excluded in order to achieve good results. However, even taking these transformations into account, there were still large differences between the two problem sets.

In the third part of this thesis, we have shown the effects of combining both the problem and the performance spaces by tackling the problem of algorithm selection. While algorithm selection is a well-studied area, most of the current state-of-the-art research has been limited to only a single problem set. We instead focused our attention on determining the generalizability of ELA for algorithm selection by examining how an algorithm selection model trained on a set of artificially generated problems performs when evaluated on a completely separate set of handmade benchmark problems.

We have shown that even though such a model can perform well when trained and evaluated on the same set of problems (either artificial or handmade, assuming all of the base problems are present in both the training and the testing set), the model performs poorly when trained on the artificial set and evaluated on the handmade set, i.e., in a generalized learning task. This further shows that ELA has issues with generalizability, and that careful attention needs to be paid when using it across different types of problems.

We then performed several additional experiments with the goal of explaining the poor performance of the generalized learning task. We first examined the optimization problems using correlation analysis. Then, we used the SHAP framework to examine the algorithm selection models and determine which landscape features they consider to be important when making their decisions. We determined that there are large differences between models that perform well and those that perform poorly in terms of feature importance, and that the generalized learning model was not able to correctly group together similar benchmark problems in the way that the well performing models were able to.

All in all, this thesis shows some of the current challenges involved in state-of-the-art analysis of optimization problem landscapes, specifically generalizing information between different sets of problems. While ELA shows promise and can produce good results within one problem set, we have shown that it has problems generalizing its results across different sets of problems and that its features can be affected by the transformations of shifting and scaling.

6.2 Hypotheses

In Chapter 1, we introduced three hypotheses that we have examined in the rest of this thesis.

Hypothesis 1 stated that the development of novel optimization algorithms in recent years does not lead to statistically significant improvement in performance indicating that the research is becoming ever more incremental. We have found evidence to support this hypothesis in Chapter 3, by showing that there is no statistically significant improvement in the performance of the winner of the 2018 CEC competitions compared to earlier years, and by showing similar results in the analysis of the BBOB benchmarks.

Hypothesis 2 stated that the problem space of optimization problems as determined by Exploratory Landscape Analysis features can be sensitive to the transformations of shifting and scaling. We have found evidence to support this hypothesis in Chapter 4 by showing that certain landscape features are affected by shifting and scaling.

Hypothesis 3 stated that exploratory landscape analysis does not represent optimization problems generally enough to use it for automated algorithm selection if the training and testing sets contain problems with large differences in design. Specifically, if the algorithm selection model is trained on an algorithmically generated set of problems and evaluated on a benchmark set designed by domain experts. We have found evidence to support this hypothesis in Chapter 5, where we have shown poor algorithm selection performance when using a model trained on algorithmically generated problems and tested on the problems from the BBOB benchmark.

6.3 Future Work

In this thesis, we have shown that generalizability between different problem sets is a large barrier to achieving good algorithm selection results. As such, it is clear that future work should focus on addressing this issue.

In terms of the performance space, this could primarily be solved by a more detailed analysis of different problem sets, particularly in terms of moving our focus outside of the CEC competitions and the BBOB workshops. Such benchmark problems are often designed to be difficult for algorithms to solve, however, they might not necessarily reflect the properties of real-world problems (Tušar, 2018). Unfortunately, the availability of real-world problems is limited, and because of this conducting large-scale studies on algorithm performance on such problems is difficult. However, if real-world problems become more available, one avenue of future research would be to see how the performance on the benchmark problems analyzed in this thesis correlates with real-life performance, and whether the development of optimization algorithms for benchmark competitions has led to an improvement in real-life performance.

In terms of the problem space, the analysis conducted in this thesis of the invariance of landscape features to shifting and scaling is fairly comprehensive but leaves room for future work. A natural extension of the work done in this thesis would be to expand the study of the invariance of landscape features to different transformations. One such example is rotation. This transformation does have an effect on the performance of some optimization algorithms, because it can change the properties of the underlying problem such as its separability. However, there do exist algorithms, such as CMA-ES, which are invariant to rotation (Hansen et al., 2011). In addition, landscape features that were

left out of our analysis due to computational complexity or the requirement for an exact problem definition to be known could be analyzed in future work.

Our algorithm selection experiments have several options for future work. The first is using different problem sets. In our experiments, we limited ourselves to only a single artificial problem generator and a single benchmark set. We would like to expand our experiments with additional problem sets by using different problem generators and examining different benchmarks, for example, the CEC benchmarks. We would also like to try out additional machine learning scenarios, for example training a model on the BBOB problems and testing it on the artificial problems. Another avenue of research would be to improve the machine learning results. In our experiments, we deliberately did not focus on tuning machine learning algorithms in order to achieve the best possible performance, since we were only interested in a relative comparison between different models. It might be possible to improve machine learning performance, either by using different models (for example neural networks) or by using techniques designed for transfer learning, such as the ones described in (Zhuang et al., 2020) that were already shown to be effective in other domains.

Finally, our work could be extended by creating a comprehensive dataset that would combine information on both the performance and the problem space by including a large and diverse collection of algorithm performance and corresponding problem landscape data. Or alternatively, to develop a framework that would allow the calculation of such data, and allow easy inclusion of additional algorithms or problems. Such a large dataset or framework would greatly ease the development and the evaluation of new algorithm selection approaches, increase the focus of the research community on generalizability, and help with the current issue of algorithm selection being focused primarily on the BBOB problem set.

References

Adenso-Diaz, B., & Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research*, *54*(1), 99–114.

Alarie, S., Audet, C., Gheribi, A. E., Kokkolaras, M., & Le Digabel, S. (2021). Two decades of blackbox optimization applications. *EURO Journal on Computational Optimization*, *9*, 100011.

Aranha, C., Camacho Villalón, C. L., Campelo, F., Dorigo, M., Ruiz, R., Sevaux, M., Sörensen, K., & Stützle, T. (2022). Metaphor-based metaheuristics, a call for action: The elephant in the room. *Swarm Intelligence*, *16*(1), 1–6.

Ardia, D., Mullen, K. M., Peterson, B. G., & Ulrich, J. (2016). *DEoptim: Differential evolution in R* [version 2.2-4]. https://CRAN.R-project.org/package=DEoptim

Atamna, A. (2015). Benchmarking ipop-cma-es-tpa and ipop-cma-es-msr on the bbob noiseless testbed. *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 1135–1142.

Auger, A., & Hansen, N. (2005). Performance evaluation of an advanced local search evolutionary algorithm. *2005 IEEE congress on evolutionary computation*, *2*, 1777–1784.

Awad, N., Ali, M., Liang, J., Qu, B., & Suganthan, P. (2016). Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective bound constrained real-parameter numerical optimization. *Technical report*. Nanyang Technological University Singapore.

Awad, N. H., Ali, M. Z., Suganthan, P. N., & Reynolds, R. G. (2016). An ensemble sinusoidal parameter adaptation incorporated with L-SHADE for solving CEC 2014 benchmark problems. *CEC*, 2958–2965.

Awad, N. H. and Ali, M. Z. and Liang, J. J. and Qu, B. Y. and Suganthan, P. N. (2016). Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective real-parameter numerical optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*.

Balaprakash, P., Birattari, M., & Stützle, T. (2007). Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. *International workshop on hybrid metaheuristics*, 108–122.

Bartz-Beielstein, T., Lasarczyk, C. W., & Preuß, M. (2005). Sequential parameter optimization. *2005 IEEE congress on evolutionary computation*, *1*, 773–780.

BBOB algorithm data. (2018). https://numbbo.github.io/data-archive/bbob/

Beachkofski, B., & Grandhi, R. (2002). Improved distributed hypercube sampling. *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 1274.

Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K., et al. (2002). A racing algorithm for configuring metaheuristics. *Gecco*, *2*(2002).

Bonnans, J.-F., Gilbert, J. C., Lemaréchal, C., & Sagastizábal, C. A. (2006). *Numerical optimization: Theoretical and practical aspects*. Springer Science & Business Media.

Bosman, P. A., Grahl, J., & Thierens, D. (2009). AMaLGaM IDEAs in noiseless black-box optimization benchmarking. *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, 2247–2254.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Buzdalov, M., Doerr, B., Doerr, C., & Vinokurov, D. (2020). Fixed-target runtime analysis. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 1295–1303.

Cenikj, G., Lang, R. D., Engelbrecht, A. P., Doerr, C., Korošec, P., & Eftimov, T. (2022). SELECTOR: Selecting a representative benchmark suite for reproducible statistical comparison. *Proceedings of the Genetic and Evolutionary Computation Conference*, 620–629.

Cheng, R., & Jin, Y. (2014). A competitive swarm optimizer for large scale optimization. *IEEE transactions on cybernetics*, *45*(2), 191–204.

Christie, L. A., Brownlee, A. E., & Woodward, J. R. (2018). Investigating benchmark correlations when comparing algorithms with parameter tuning. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 209–210.

Deb, K., & Goyal, M. (1996). A combined genetic adaptive search (GeneAS) for engineering design. *Computer Science and informatics*, *26*, 30–45.

de Lacerda, M. G. P., de Araujo Pessoa, L. F., de Lima Neto, F. B., Ludermir, T. B., & Kuchen, H. (2021). A systematic literature review on general parameter control for evolutionary and swarm-based algorithms. *Swarm and Evolutionary Computation*, *60*, 100777.

Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*(1), 3–18.

Doerr, B., Fouz, M., Schmidt, M., & Wahlstrom, M. (2009). BBOB: nelder-mead with resize and halfruns. *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, 2239–2246.

Duro, J. A., & de Oliveira, J. V. (2008). Particle swarm optimization applied to the chess game. *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 3702–3709.

Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 39–43.

Eftimov, T., Korošec, P., & Seljak, B. Disadvantages of statistical comparison of stochastic optimization algorithms. In: 2016, 105–118.

Eftimov, T., Popovski, G., Renau, Q., Korošec, P., & Doerr, C. (2020). Linear matrix factorization embeddings for single-objective optimization landscapes. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 775–782.

Eftimov, T., & Korošec, P. (2018). The impact of statistics for benchmarking in evolutionary computation research. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 1329–1336.

Eftimov, T., Korošec, P., & Koroušić Seljak, B. (2017). A novel approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics. *Information Sciences*, *417*, 186–215.

Eftimov, T., Korošec, P., & Seljak, B. K. (2017). A novel approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics. *Information Sciences*, *417*, 186–215.

Eftimov, T., Petelin, G., Hribar, R., Popovski, G., Škvorc, U., & Korošec, P. (2020). Deep statistics: More robust performance statistics for single-objective optimization benchmarking. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 5–6.

Eiben, Á. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, *3*(2), 124–141.

Elsayed, S. M., Sarker, R. A., Essam, D. L., & Hamza, N. M. (2014). Testing united multi-operator evolutionary algorithms on the CEC 2014 real-parameter numerical optimization. *Evolutionary Computation (CEC), 2014 IEEE Congress on*, 1650–1657.

Fialho, Á., Schoenauer, M., & Sebag, M. (2010). Fitness-AUC bandit adaptive strategy selection vs. the probability matching one within differential evolution: An empirical comparison on the BBOB-2010 noiseless testbed. *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, 1535–1542.

Flamm, C., Hofacker, I. L., Stadler, P. F., & Wolfinger, M. T. (2002). Barrier trees of degenerate landscapes.

Friedman, J., Hastie, T., Tibshirani, R., et al. (2001). *The elements of statistical learning* (Vol. 1). Springer series in statistics New York.

García, S., Molina, D., Lozano, M., & Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization. *Journal of Heuristics*, *15*(6), 617–644.

García-Martínez, C., & Lozano, M. (2009). A continuous variable neighbourhood search based on specialised EAs: Application to the noiseless BBO-benchmark 2009. *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, 2287–2294.

Halton, J. H. (1960). On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, *2*(1), 84–90.

Hansen, N. (2009). Benchmarking a bi-population CMA-ES on the BBOB-2009 function testbed. *Proceedings of the 11th annual conference companion on genetic and evolutionary computation conference: late breaking papers*, 2389–2396.

Hansen, N., Auger, A., Brockhoff, D., & Tušar, T. (2022). Anytime performance assessment in blackbox optimization benchmarking. *IEEE Transactions on Evolutionary Computation*, *26*(6), 1293–1305.

Hansen, N., Auger, A., Finck, S., & Ros, R. (2012). Real-parameter black-box optimization benchmarking: Experimental setup. *Orsay, France: Université Paris Sud, Institut National de Recherche en Informatique et en Automatique (INRIA) Futurs, Équipe TAO, Tech. Rep.*

Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., & Brockhoff, D. (2021). COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, *36*(1), 114–144.

Hansen, N., Finck, S., Ros, R., & Auger, A. (2009). Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. *[Research Report] RR-6829, INRIA. 2009. inria-00362633v2.*

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, *9*(2), 159–195.

Hansen, N., & Ros, R. (2010). Benchmarking a weighted negative covariance matrix update on the BBOB-2010 noiseless testbed. *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, 1673–1680.

Hansen, N., Ros, R., Mauny, N., Schoenauer, M., & Auger, A. (2011). Impacts of invariance in search: When CMA-ES and PSO face ill-conditioned and non-separable problems. *Applied Soft Computing*, *11*(8), 5755–5769.

Huang, C., Li, Y., & Yao, X. (2019). A survey of automatic parameter tuning methods for metaheuristics. *IEEE transactions on evolutionary computation*, *24*(2), 201–216.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. *International conference on learning and intelligent optimization*, 507–523.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*, 267–306.

IEEE standard for floating-point arithmetic. (2008). *IEEE Std 754-2008*, 1–70.

Jankovic, A., Eftimov, T., & Doerr, C. (2021). Towards feature-based performance regression using trajectory data. *Applications of Evolutionary Computation (EvoApplications 2021)*, *12694*, 601–617.

Jankovic, A., Popovski, G., Eftimov, T., & Doerr, C. (2021). The impact of hyper-parameter tuning for landscape-aware performance regression and algorithm selection. *arXiv preprint arXiv:2104.09272*.

Jansen, T. (2013). *Analyzing evolutionary algorithms: The computer science perspective*. Springer Science & Business Media.

Jansen, T., & Zarges, C. (2014). Performance analysis of randomised search heuristics operating with a fixed budget. *Theoretical Computer Science*, *545*, 39–58.

Karaboga, D. (2005). *An idea based on honey bee swarm for numerical optimization* (tech. rep.). Citeseer.

Karafotias, G., Hoogendoorn, M., & Eiben, A. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, *19*(2), 167–187.

Kerschke, P., Hoos, H. H., Neumann, F., & Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, *27*(1), 3–45.

Kerschke, P., Preuss, M., Hernández, C., Schütze, O., Sun, J.-Q., Grimme, C., Rudolph, G., Bischl, B., & Trautmann, H. (2014). Cell mapping techniques for exploratory landscape analysis. *EVOLVE-A bridge between probability, set oriented numerics, and evolutionary computation v* (pp. 115–131). Springer.

Kerschke, P., Preuss, M., Wessing, S., & Trautmann, H. (2015). Detecting funnel structures by means of exploratory landscape analysis. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 265–272.

Kerschke, P., & Trautmann, H. (2019a). Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary computation*, *27*(1), 99–127.

Kerschke, P., & Trautmann, H. (2019b). Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the r-package flacco. In N. Bauer, K. Ickstadt, K. Lübke, G. Szepannek, H. Trautmann, & M. Vichi (Eds.), *Applications in statistical computing – from music data analysis to industrial quality improvement* (pp. 93–123). Springer. https://link.springer.com/chapter/10.1007/978-3-030-25147-5_7

Kumar, A., Misra, R. K., & Singh, D. (2017). Improving the local search capability of effective butterfly optimizer using covariance matrix adapted retreat phase. *Evolutionary Computation (CEC), 2017 IEEE Congress on*, 1835–1842.

Lacroix, B., Christie, L. A., & McCall, J. A. (2017). Interpolated continuous optimisation problems with tunable landscape features. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 169–170.

Lacroix, B., & McCall, J. (2019). Limitations of benchmark sets and landscape features for algorithm selection and performance prediction. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 261–262.

Lang, R. D., & Engelbrecht, A. P. (2021). An exploratory landscape analysis-based benchmark suite. *Algorithms*, *14*(3), 78.

LaTorre, A., Muelas, S., & Peña, J. M. (2010). Benchmarking a mos-based algorithm on the BBOB-2010 noiseless function testbed. *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, 1649–1656.

Lex, A., Gehlenborg, N., Strobelt, H., Vuillemot, R., & Pfister, H. (2014). Upset: Visualization of intersecting sets. *IEEE transactions on visualization and computer graphics*, *20*(12), 1983–1992.

Liang, J. J., Qu, B. Y., & Suganthan, P. N. (2013). Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*.

Liang, J. J., Qu, B. Y., Suganthan, P. N., & Chen, Q. (2014). Problem definitions and evaluation criteria for the CEC 2015 competition on learning-based real-parameter single objective optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*.

Liang, J. J., Qu, B. Y., Suganthan, P. N., & Hernández-Díaz, A. G. (2013). Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*.

Liang, J.-J., Suganthan, P. N., & Deb, K. (2021). *Github - ponnuthurai nagaratnam suganthan*. https://github.com/P-N-Suganthan

Liaw, A., & Wiener, M. (2002). Classification and regression by randomforest. *R News*, *2*(3), 18–22. https://CRAN.R-project.org/doc/Rnews/

Lindauer, M., Bergdoll, R.-D., & Hutter, F. (2016). An empirical study of per-instance algorithm scheduling. *Learning and Intelligent Optimization: 10th International Conference, LION 10, Ischia, Italy, May 29–June 1, 2016, Revised Selected Papers 10*, 253–259.

Loshchilov, I. (2013). CMA-ES with restarts for solving CEC 2013 benchmark problems. *Evolutionary Computation (CEC), 2013 IEEE Congress on*, 369–376.

Loshchilov, I., Schoenauer, M., & Sebag, M. (2012a). Black-box optimization benchmarking of IPOP-saACM-ES and BIPOP-saACM-ES on the BBOB-2012 noiseless testbed. *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, 175–182.

Loshchilov, I., Schoenauer, M., & Sebag, M. (2013). Bi-population CMA-ES agorithms with surrogate models and line searches. *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, 1177–1184.

Loshchilov, I., Schoenauer, M., & Sebag, M. (2012b). Black-box optimization benchmarking of NIPOP-ACMA-ES and NBIPOP-ACMA-ES on the BBOB-2012 noiseless

testbed. *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, 269–276.

Loshchilov, I., Schoenauer, M., & Sebag, M. (2012c). Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, 321–328.

Lunacek, M., & Whitley, D. (2006). The dispersion metric and the CMA evolution strategy. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 477–484.

Lunacek, M., Whitley, D., & Sutton, A. (2008). The impact of global structure on search. *International Conference on Parallel Problem Solving from Nature*, 498–507.

Lundberg, S. M. (2018). Shap.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S.-I. (2020). From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, *2*(1), 2522–5839.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Proceedings of the 31st international conference on neural information processing systems*, 4768–4777.

Matoušek, J. (1998). On the l2-discrepancy for anchored boxes. *Journal of Complexity*, *14*(4), 527–556.

McDermott, J. (2020). When and why metaheuristics researchers can ignore "no free lunch" theorems. *SN Computer Science*, *1*(1), 1–18.

McKay, M. D., Beckman, R. J., & Conover, W. J. (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *42*(1), 55–61.

Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., & Rudolph, G. (2011). Exploratory landscape analysis. *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 829–836.

Mersmann, O., Preuss, M., & Trautmann, H. (2010). Benchmarking evolutionary algorithms: Towards exploratory landscape analysis. *International Conference on Parallel Problem Solving from Nature*, 73–82.

Molina, D., LaTorre, A., & Herrera, F. (2018). An insight into bio-inspired and evolutionary algorithms for global optimization: Review, analysis, and lessons learnt over a decade of competitions. *Cognitive Computation*, *10*, 517–544.

Muñoz, M. A., Kirley, M., & Halgamuge, S. K. (2014). Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE Transactions on Evolutionary Computation*, *19*(1), 74–87.

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

Muñoz, M. A., & Smith-Miles, K. A. (2017). Performance analysis of continuous black-box optimization algorithms via footprints in instance space. *Evolutionary computation*, *25*(4), 529–554.

Netrapalli, P. (2019). Stochastic Gradient Descent and Its Variants in Machine Learning. *Journal of the Indian Institute of Science*, *99*(2), 201–213.

Nikolikj, A., Trajanov, R., Cenikj, G., Korošec, P., & Eftimov, T. (2022). Identifying minimal set of exploratory landscape analysis features for reliable algorithm performance prediction. *2022 IEEE Congress on Evolutionary Computation (CEC)*, 1–8.

Nishida, K., & Akimoto, Y. (2018). Benchmarking the PSA-CMA-ES on the BBOB noiseless testbed. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 1529–1536.

Numerical characteristics of the machine [https://stat.ethz.ch/R-manual/R-devel/library/base/html/zMachine.html]. (2022).

Oliveira, C., Aleti, A., Grunske, L., & Smith-Miles, K. (2018). Mapping the Effectiveness of Automated Test Suite Generation Techniques. *IEEE Transactions on Reliability*, *67*(3), 771–785.

Pošík, P., & Klemš, V. (2012a). Benchmarking the differential evolution with adaptive encoding on noiseless functions. *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, 189–196.

Pošík, P., & Klemš, V. (2012b). Jade, an adaptive differential evolution algorithm, benchmarked on the bbob noiseless testbed. *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, 197–204.

Price, K., Storn, R. M., & Lampinen, J. A. (2006). *Differential evolution: A practical approach to global optimization*. Springer.

Price, K. V. (1997). Differential evolution vs. the functions of the 2^{nd} ICEO. *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, 153–157.

Rastrigin, L. A. (1974). Systems of extremal control. *Nauka*.

Renau, Q., Doerr, C., Dreo, J., & Doerr, B. (2020). Exploratory landscape analysis is strongly sensitive to the sampling strategy. *International Conference on Parallel Problem Solving from Nature*, 139–153.

Rice, J. R. (1976). The algorithm selection problem. *Advances in computers* (pp. 65–118). Elsevier.

Ros, R. (2009). Benchmarking sep-CMA-ES on the BBOB-2009 function testbed. *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, 2435–2440.

Rosenbrock, H. H. (1960). An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, *3*(3), 175–184.

Saltelli, A., Annoni, P., Azzini, I., Campolongo, F., Ratto, M., & Tarantola, S. (2010). Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index. *Computer physics communications*, *181*(2), 259–270.

Schumacher, C., Vose, M. D., Whitley, L. D., et al. (2001). The no free lunch and problem description length. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 565–570.

Sheskin, D. J. (2003). *Handbook of parametric and nonparametric statistical procedures*. Chapman; Hall/CRC.

Škvorc, U., Eftimov, T., & Korošec, P. (2020). Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis. *Applied Soft Computing*, *90*, 106138.

Škvorc, U., Eftimov, T., & Korošec, P. (2021). The effect of sampling methods on the invariance to function transformations when using exploratory landscape analysis. *2021 IEEE Congress on Evolutionary Computation (CEC)*, 1139–1146.

Škvorc, U., Eftimov, T., & Korošec, P. (2022). A comprehensive analysis of the invariance of exploratory landscape analysis features to function transformations. *2022 IEEE Congress on Evolutionary Computation (CEC)*, 1–8.

Smith-Miles, K., Baatar, D., Wreford, B., & Lewis, R. (2014). Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, *45*, 12–24.

Smith-Miles, K., & Bowly, S. (2015). Generating new test instances by evolving in instance space. *Computers & Operations Research*, *63*, 102–113.

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

Song, A., Wu, G., Mallipeldi, R., & P. N. Suganthan. (2019). *Comparison of results in 2019 on CEC competition on constrained parameter optimization* (tech. rep.). https://github.com/P-N-Suganthan/CEC2017/blob/master/Comparison%5C%20of%5C%20Results%5C%20in%5C%202019%5C%20on%5C%20CEC%5C%20Competition%5C%20on%5C%20Constrained%5C%20RealParameter%5C%20Optimization-2017.pdf

Stork, J., Eiben, A. E., & Bartz-Beielstein, T. (2022). A new taxonomy of global optimization algorithms. *Natural Computing*, *21*(2), 219–242.

Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, *11*(4), 341–359.

Suganthan, P. N., Ali, M. Z., & Awad, N. H. (2016). *CEC 2016 Special Session on Single Objective Numerical Optimization Single parameter-operator set based case*. https://github.com/P-N-Suganthan/CEC2016/

Tanabe, R., & Fukunaga, A. S. (2014). Improving the search performance of SHADE using linear population size reduction. *Evolutionary Computation (CEC), 2014 IEEE Congress on*, 1658–1665.

Tian, Y., Peng, S., Zhang, X., Rodemann, T., Tan, K. C., & Jin, Y. (2020). A recommender system for metaheuristic algorithms for continuous optimization based on deep recurrent neural networks. *IEEE Transactions on Artificial Intelligence*, *1*(1), 5–18.

Törn, A., Ali, M. M., & Viitanen, S. (1999). Stochastic global optimization: Problem classes and solution techniques. *Journal of Global Optimization*, *14*(4), 437–447.

Trajanov, R., Dimeski, S., Popovski, M., Korošec, P., & Eftimov, T. (2021). Explainable landscape-aware optimization performance prediction. *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 01–08.

Tušar, T. (2018). On using real-world problems for benchmarking multiobjective optimization algorithms. *Proceedings of the International Conference on High-Performance Optimization in Industry, HPOI 2018, 21st International Multiconference Information Society, IS 2018*, *500*.

Van Laarhoven, P. J., & Aarts, E. H. (1987). Simulated annealing. *Simulated annealing: Theory and applications* (pp. 7–15). Springer.

Vrugt, J. A., & Robinson, B. A. (2007). Improved evolutionary optimization from genetically adaptive multimethod search. *Proceedings of the National Academy of Sciences*, *104*(3), 708–711.

Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, *2*(1-3), 37–52.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, *1*(1), 67–82.

Xu, T., He, J., & Shang, C. (2019). Helper and equivalent objective different evolution for constrained optimisation. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 9–10.

Yamaguchi, T., & Akimoto, Y. (2017). Benchmarking the novel CMA-ES restart strategy using the search history on the BBOB noiseless testbed. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 1780–1787.

Yang, X.-S. (2014). *Nature-inspired optimization algorithms* (1st). Elsevier Science Publishers B. V.

Yao, X., Liu, Y., & Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, *3*(2), 82–102.

Ying, X. (2019). An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, *1168*(2), 022022.

Yuan, Z., Montes de Oca, M. A., Birattari, M., & Stützle, T. (2012). Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence*, *6*(1), 49–75.

Yuan, Z., Stützle, T., Montes de Oca, M. A., Lau, H. C., & Birattari, M. (2013). An analysis of post-selection in automatic configuration. *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 1557–1564.

Zabinsky, Z. B. (2010). Random search algorithms. *Wiley encyclopedia of operations research and management science*.

Zamuda, A. (2017). Adaptive constraint handling and success history differential evolution for CEC 2017 constrained real-parameter optimization. *2017 IEEE Congress on Evolutionary Computation (CEC)*, 2443–2450.

Zhang, G., & Shi, Y. (2018). Hybrid sampling evolution strategy for solving single objective bound constrained problems. *2018 IEEE Congress on Evolutionary Computation (CEC)*, 1–7.

Zhang, Y.-W., & Halgamuge, S. K. (2019). Similarity of continuous optimization problems from the algorithm performance perspective. *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2949–2957.

Zhu, J., Bandler, J. W., Nikolova, N. K., & Koziel, S. (2007). Antenna optimization through space mapping. *IEEE Transactions on Antennas and Propagation*, *55*(3), 651–658.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, *109*(1), 43–76.

Bibliography

Publications Related to the Thesis

Journal Articles

Škvorc, U., Eftimov, T., & Korošec, P. (2020a). Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis. *Applied Soft Computing*, *90*, 106138.

Škvorc, U., Eftimov, T., & Korošec, P. (2022b). Transfer learning analysis of multi-class classification for landscape-aware algorithm selection. *Mathematics*, *10*(3). https://www.mdpi.com/2227-7390/10/3/432

Conference Papers

Škvorc, U., Eftimov, T., & Korošec, P. (2019a). CEC real-parameter optimization competitions: Progress from 2013 to 2018. *2019 IEEE Congress on Evolutionary Computation (CEC)*, 3126–3133.

Škvorc, U., Eftimov, T., & Korošec, P. (2021a). A complementarity analysis of the COCO benchmark problems and artificially generated problems. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 215–216.

Škvorc, U., Eftimov, T., & Korošec, P. (2021b). The effect of sampling methods on the invariance to function transformations when using exploratory landscape analysis. *2021 IEEE Congress on Evolutionary Computation (CEC)*, 1139–1146.

Other Publications

Conference Papers

Eftimov, T., Hribar, R., Škvorc, U., Popovski, G., Petelin, G., & Korošec, P. (2020). Performviz: A machine learning approach to visualize and understand the performance of single-objective optimization algorithms. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 7–8.

Eftimov, T., Petelin, G., Hribar, R., Popovski, G., Škvorc, U., & Korošec, P. (2020). Deep statistics: More robust performance statistics for single-objective optimization benchmarking. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 5–6.

Biography

Urban Škvorc was born on April 9^{th}, 1993 in Ljubljana, Slovenia, where he attended primary and secondary school. In 2015, he was awarded a Bachelor's degree, and in 2017 a Master's degree from the Faculty of Computer and Information Science of the University of Ljubljana for his Master's thesis titled "Runtime hardware reconfiguration". In 2018, he enrolled into the doctoral program Information and communication technologies at the Jožef Stefan International Postgraduate School under a young researcher grant awarded by the Slovenian Research Agency.

His research interests include optimization, benchmarking, and exploratory landscape analysis, and machine learning. Specifically, he has focused on examining the reliability and transferability of knowledge obtained using exploratory landscape analysis for the task of algorithm selection. During his PhD studies, he has presented his work at several leading conferences in his field, specifically the IEEE Congress on Evolutionary Computation (IEEE CEC) and the ACM Genetic and Evolutionary Computation Conference (GECCO).