# PROBLEM LANDSCAPE AND SEARCH TRAJECTORY FEATURES FOR AUTOMATED ALGORITHM SELECTION IN CONTINUOUS SINGLE-OBJECTIVE OPTIMIZATION

Gjorgjina Cenikj

**Doctoral Dissertation**
**Jožef Stefan International Postgraduate School**
**Ljubljana, Slovenia**

**Supervisor:** Asst. Prof. Dr. Tome Eftimov, Jožef Stefan Institute, Ljubljana, Slovenia
**Co-Supervisor:** Prof. Dr. Barbara Koroušić Seljak, Jožef Stefan Institute, Ljubljana, Slovenia

**Evaluation Board:**
Prof. Dr. Ljupčo Todorovski, Chair, Jožef Stefan International Postgraduate School and Faculty of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia
Dr. Carola Doerr, Member, CNRS and Sorbonne University, Paris, France
Asst. Prof. Dr. Niki Van Stein, Member, LIACS, Leiden University, Leiden, Netherlands

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL




Gjorgjina Cenikj

# PROBLEM LANDSCAPE AND SEARCH TRAJECTORY FEATURES FOR AUTOMATED ALGORITHM SELECTION IN CONTINUOUS SINGLE-OBJECTIVE OPTIMIZATION

**Doctoral Dissertation**

# ZNAČILKE PROBLEMSKE POKRAJINE IN ISKALNIH TRAJEKTORIJ ZA AVTOMATIZIRANO IZBIRO ALGORITMOV V ZVEZNI ENOKRITERIJSKI OPTIMIZACIJI

**Doktorska disertacija**

**Supervisor:** Asst. Prof. Dr. Tome Eftimov

**Co-Supervisor:** Prof. Dr. Barbara Koroušić Seljak

Ljubljana, Slovenia, October 2025

# Acknowledgments

Firstly, I would like to express my deepest gratitude to my supervisors, Tome Eftimov and Prof. Dr. Barbara Koroušić, for their unwavering support, trust, and encouragement throughout this PhD journey. Thank you for granting me the creative freedom to explore, for guiding me with clarity and patience, and for always keeping your doors open when I needed advice. Your mentorship has shaped me both as a researcher and as a person.

I would also like to thank the committee, Prof. Dr. Ljupčo Todorovski, Dr. Carola Doerr, and Asst. Prof. Dr. Niki Van Stein, for their valuable feedback and insightful comments, which have helped refine and strengthen this work.

On a more personal note, I would like to express my heartfelt appreciation to the people who carried me through this journey.

To Tome. For your unwavering enthusiasm, support, and encouragement. For making me feel that we can do anything, even when the odds seemed impossible. This thesis would not happen without you.

To Gašper. For listening to every silly little crisis. For understanding me. For embracing my quirks and craziness. For being my partner in every sense of the word.

To my brother. For your silliness that lit up my dullest days. For knowing my punchline before I start the joke. For always having my back.

To my dad. For pushing me to be the best version of myself. For believing I would do a PhD even when I insisted there was no chance. For the quiet sacrifices and unwavering support.

To my mom. For the countless little things. For carrying the weight of the world so I would not need to. For your love, worry and sacrifice.

To my extended family, friends and colleagues. For sharing this journey with me in laughter and in frustration. For reminding me that research is not only about ideas and results, but also about community, friendship, and support.

# Abstract

Black-box single-objective optimization (SOO) plays a critical role in solving diverse problems across engineering, machine learning, and scientific computation. Despite extensive research in this area, no single optimization algorithm performs best across all problem instances. This challenge has led to increased interest in automated algorithm selection (AS), which aims to match problem instances with the most effective algorithms based on descriptive problem landscape features.

This dissertation investigates three key limitations in existing AS methodologies for continuous SOO. First, it addresses the limitations of traditional landscape features, such as those from Exploratory Landscape Analysis (ELA), by exploring alternative representations learned through deep learning and topological data analysis. Second, it highlights the variability and inconsistencies in AS evaluation protocols—particularly the influence of different benchmarking strategies on model performance and interpretability. Third, it introduces trajectory-based features that capture dynamic interactions between optimization algorithms and problem landscapes, offering a complementary perspective beyond static landscape descriptors.

Contrary to expectations, empirical findings reveal that the newly proposed features, while capturing novel aspects of problem complexity, do not consistently outperform ELA features in AS tasks. Moreover, models built on both traditional and novel feature sets demonstrate limited generalizability, particularly when evaluated under more challenging benchmark settings or across diverse problem domains. These results underscore the persistent difficulty of building robust AS models and highlight the need for improved benchmarking standards and more expressive, generalizable feature representations.

Through the systematic evaluation of feature types and AS protocols, this dissertation contributes a comprehensive analysis of the current limitations and future potential of algorithm selection in continuous SOO.

# Povzetek

Zvezna enokriterijska optimizacija ima ključno vlogo pri reševanju raznolikih aplikativnih problemov na področjih industrije, strojnega učenja in računalništva. Kljub obsežnim raziskavam na področju optimizacije, noben algoritem ne izkazuje univerzalne učinkovitosti za vse tipe problemov. Ta omejitev je spodbudila intenzivno raziskovalno zanimanje za avtomatizirano izbiro algoritmov, katere osrednji cilj je vzpostaviti ustrezno povezavo med posameznimi problemi in učinkovitostjo algoritmov na podlagi opisnih značilk problemskih pokrajin.

Ta disertacija obravnava tri temeljne omejitve obstoječih metodologij za avtomatizirano izbiro algoritmov pri zvezni enociljni optimizaciji. Prvič, naslavlja omejitve tradicionalnih značilk problemskih pokrajin, kot so tiste iz Exploratory Landscape Analysis, z raziskovanjem alternativnih predstavitev, pridobljenih z globokim učenjem in topološko analizo podatkov. Drugič, opozarja na variabilnost in nedoslednosti v evalvacijskih protokolih avtomatizirane izbire algoritmov, pri čemer posebej izpostavlja vpliv različnih strategij testiranja na uspešnost in razložljivost modelov. Tretjič, uvaja značilke, utemeljene na poteh optimizacijskih algoritmov, ki zajemajo dinamične interakcije med algoritmi in problemskimi pokrajinami ter ponujajo dopolnilno perspektivo onkraj statičnih deskriptorjev.

V nasprotju s pričakovanji empirični rezultati kažejo, da novo predlagane značilke, čeprav zajemajo dodatne vidike lastnosti problemov, ne izboljšajo točnosti izbire algoritmov v primerjavi z obstoječimi značilkami. Poleg tega modeli, zasnovani tako na tradicionalnih kot tudi na novih naborih značilk, izkazujejo omejeno splošno uporabnost, zlasti kadar so ovrednoteni v zahtevnejših testnih pogojih ali na raznolikih problemskih domenah. Ti rezultati poudarjajo trajno zahtevnost razvoja robustnih modelov za izbiro algoritmov ter potrebo po izboljšanih evalvacijskih standardih in bolj izraznih, splošno uporabnih značilkah.

S sistematično evalvacijo različnih vrst značilk za avtomatizirano izbiro algoritmov disertacija ponuja poglobljeno analizo delovanja značilk in modelov, ki jih ti uporabljajo. Prav tako izpostavlja obstoječe omejitve pri izbiri algoritmov ter nakazuje nadaljnji razvojni potencial tega področja v zvezni enociljni optimizaciji.

# Contents

**List of Figures** **xv**

**List of Tables** **xix**

**Abbreviations** **xxi**

**1 Introduction** **1**
1.1 Research Goals . . . 3
1.2 Hypotheses . . . 4
1.3 Scientific Contributions . . . 4
1.4 Thesis Structure . . . 5

**2 Background and related work** **7**
2.1 Machine Learning Pipeline using Optimization Features . . . 7
2.2 Problem Portfolios . . . 10
2.2.1 Benchmark suites . . . 10
2.2.2 Problem generators . . . 11
2.3 Problem Landscape Features . . . 12
2.3.1 Exploratory landscape analysis . . . 13
2.3.2 Topological landscape analysis . . . 15
2.3.3 Deep learning-based approaches . . . 16
2.3.4 Advantages vs. disadvantages of problem landscape features . . . 18
2.4 Algorithm Features . . . 19
2.4.1 Algorithm features based on source code . . . 19
2.4.2 Advantages vs. disadvantages of algorithm features . . . 19
2.5 High-Level Problem-Algorithm Interaction Features . . . 19
2.5.1 Algorithm features based on performance . . . 19
2.5.2 Algorithm features based on problem landscape features used by performance prediction models . . . 20
2.5.3 Algorithm features using graph embeddings . . . 20
2.5.4 Features based on configuration settings for modular algorithm frameworks . . . 21
2.5.5 Advantages vs. disadvantages of high-Level problem-algorithm interaction features . . . 21
2.6 Trajectory-based Features that Capture Problem-Algorithm Interaction . . . 23
2.6.1 Trajectory-based features based on internal algorithm parameters . . . 23
2.6.2 Trajectory-based ELA features . . . 23
2.6.3 DynamoRep features . . . 24
2.6.4 Opt2Vec features . . . 24
2.6.5 Iterative-based ELA features . . . 25
2.6.6 Local optima networks . . . 25

2.6.7 Probing trajectories . . . 25
2.6.8 ClustOpt . . . 26
2.6.9 Advantages vs. disadvantages of trajectory features. . . . 26
2.7 Machine Learning Studies that Utilized Optimization Features . . . 27

**3 Problem Landscape Features Based On Transformer Models 29**
3.1 TransOpt . . . 29
3.1.1 Methodology . . . 30
3.1.2 Results . . . 31
3.2 TransOptAS . . . 34
3.2.1 Methodology . . . 35
3.2.2 Results . . . 41
3.3 Conclusion . . . 47

**4 Algorithm Selection Generalizability 49**
4.1 A Cross-Benchmark Examination of Feature-based Algorithm Selector Generalization in Continuous Single-Objective Optimization . . . 51
4.1.1 Methodology . . . 51
4.1.2 Results . . . 55
4.1.3 Discussion . . . 68
4.2 Landscape Features in Single-Objective Continuous Optimization: Have We Hit a Wall in Algorithm Selection Generalization? . . . 70
4.2.1 Methodology . . . 70
4.2.2 Results . . . 75
4.2.3 Discussion . . . 89
4.3 Conclusion . . . 91

**5 Trajectory Features 93**
5.1 DynamoRep Representations . . . 95
5.2 Experimental Design . . . 95
5.2.1 Problem portfolio . . . 96
5.2.2 Algorithm portfolio . . . 96
5.2.3 DynamoRep feature calculation . . . 96
5.2.4 Evaluation tasks . . . 97
5.2.5 Comparison to ELA features . . . 99
5.3 Computation Time of the DynamoRep and Iteration-based ELA Features . 100
5.4 Algorithm Selection Results . . . 101
5.4.1 Algorithm selection with limited feature calculation budget . . . 102
5.4.2 Feature importance analysis . . . 104
5.5 Problem Classification Results . . . 106
5.5.1 Problem classification with limited feature calculation budgets . . . 106
5.5.2 Comparison to iteration-based ELA features . . . 107
5.5.3 Comparison to static ELA features . . . 108
5.5.4 Feature importance analysis . . . 109
5.6 Algorithm Classification Results . . . 110
5.6.1 Algorithm classification with limited feature calculation budgets . . . 110
5.6.2 Comparison to iteration-based ELA features . . . 111
5.6.3 Feature importance analysis . . . 111
5.7 Conclusion . . . 113

**6 Discussion and Conclusion 115**

6.1 Discussion . . . . . 115
6.1.1 Discussion of Hypotheses . . . . . 115
6.1.2 Discussion of Scientific Contributions . . . . . 116
6.2 Conclusion . . . . . 117

**References** **121**

**Bibliography** **135**

**Biography** **137**

# List of Figures

Figure 2.1: A typical machine learning pipeline using optimization features. . . . . 8
Figure 2.2: The general pipeline of calculating problem landscape features. . . . . 12
Figure 2.3: The general pipeline of calculating problem landscape features. . . . . 20
Figure 2.4: The general pipeline of calculating problem-algorithm trajectory features. 23

Figure 3.1: Model architecture with example inputs being samples of two-dimensional optimization problems. . . . . 32
Figure 3.2: Comparison of the accuracy obtained by the transformer model to the ELA features for the classification of problem instances into the 24 classes of problems in the BBOB benchmark . . . . . 33
Figure 3.3: Confusion matrix for the classification of problem instances of dimension 3 into the 24 classes of problems in the BBOB benchmark. . . . . 34
Figure 3.4: Confusion matrix for the classification of problem instances of dimension 20 into the 24 classes of problems in the BBOB benchmark. . . . . 35
Figure 3.5: Model architecture with example inputs being samples of two-dimensional optimization problems. . . . . 40
Figure 3.6: Mean algorithm scores for the DE, GA, ES and PSO algorithm portfolio on the 5,000 randomly generated problems in dimension 3 and 10, expressed in terms of the algorithm performance metric defined in Subsection 3.2.1.4. . . . . 42
Figure 3.7: Mean algorithm scores for the PSO configuration portfolio on the 5,000 randomly generated problems in dimension 3 and 10, expressed in terms of the algorithm performance metric defined in Subsection 3.2.1.4. . . . 43
Figure 3.8: Losses achieved by the dummy model, the transformer-based model and the ELA-based model for the DE-GA-ES-PSO algorithm portfolio, expressed in terms of the AS loss metric defined in Subsection 3.2.1.6 (the metric is supposed to be maximized, so higher value is better). . . 44
Figure 3.9: Losses achieved by the dummy model, the transformer-based model and the ELA-based model for the PSO configuration portfolio, expressed in terms of the AS loss metric defined in Subsection 3.2.1.6 (the metric is supposed to be maximized, so higher value is better). . . . . 45
Figure 3.10: Scatterplot depicting the loss (defined in Subsection 3.2.1.6) obtained with the ELA features and the transformer models, for different $3d$ problem instances, for the DE-GA-ES-PSO algorithm portfolio, a budget of 50 iterations. . . . . 46

Figure 4.1: Mean score of each optimization algorithm on $3d$ and $10d$ benchmarks, for budgets of 10, 30 and 50 iterations. . . . . 56
Figure 4.2: Pairwise ranking score and loss achieved by the models for $3d$ problems. 58
Figure 4.3: Summary results in terms of the loss metric. . . . . 60

Figure 4.4: Distribution of the values of the top most important ELA features for the AS trained for 3$d$ problems and a budget of 50 iterations. . . . . 62
Figure 4.5: Distribution of the values of the top most important TransOpt features for the AS trained for 3$d$ problems and a budget of 50 iterations. . . . . 63
Figure 4.6: Number of problem instances from each benchmark in the clusters generated using ELA features with a sample size of 100$d$. . . . . 64
Figure 4.7: Number of problem instances from each benchmark in the clusters generated using TransOpt features with a sample size of 100$d$. . . . . 65
Figure 4.8: Performance alignment of the first instance of the first 3$d$ BBOB problem class and its 10 most similar problem instances from the ZIGZAG benchmark, for a budget of 30 iterations. . . . . 66
Figure 4.9: Performance alignment of the first instance of the fifth 3$d$ BBOB problem class and its 10 most similar problem instances from the ZIGZAG benchmark, for a budget of 30 iterations. . . . . 67
Figure 4.10: Performance alignment of the first instance of each of the 3$d$ BBOB problem classes to the ZIGZAG benchmark, and loss achieved on these instances when the model is trained on the ZIGZAG benchmark. . . . . 69
Figure 4.11: Distribution of the normalized precision scores in each algorithm portfolio. A lower normalized precision signifies a better performance of an algorithm inside a portfolio. . . . . 76
Figure 4.12: Spearman correlation clustermap between pairs of features from different feature groups. . . . . 78
Figure 4.13: Spearman correlation clustermap between the *ela_scaled* features and the other feature groups. . . . . 79
Figure 4.14: Cosine similarity of problem pairs, calculated with different feature sets. The first feature name in the title of each subplot refers to the horizontal axis, while the second feature name refers to the vertical axis. . . . . 81
Figure 4.15: Distribution of feature and performance similarities for the 2DE+2PSO portfolio for 1,000,000 problem pairs. . . . . 83
Figure 4.16: Distribution of Spearman correlation coefficient of individual features within feature groups with performance of individual algorithms within the 2DE+2PSO algorithm portfolio. . . . . 85
Figure 4.17: AS performance obtained for the different feature groups and algorithm portfolios in the instance split evaluation. . . . . 86
Figure 4.18: AS performance obtained for the different feature groups and algorithm portfolios in the random split evaluation. . . . . 87
Figure 4.19: AS performance obtained for the different feature groups and algorithm portfolios in the problem combination split evaluation. . . . . 88
Figure 4.20: AS performance obtained for the different feature groups and algorithm portfolios in the problem split evaluation. . . . . 89

Figure 5.1: Mean time in seconds needed to calculate the DynamoRep features and the iteration-based ELA features for a single algorithm trajectory consisting of 30 iterations. . . . . 101
Figure 5.2: AS results using the instance-split evaluation setting for 5$d$ problems. . 102
Figure 5.3: AS results using the problem-split evaluation setting for the BBOB problems in dimension 5. . . . . 103
Figure 5.4: Top 20 DynamoRep features with highest feature importance assigned by the RF model for the AS task in the instance-split evaluation setting. 105
Figure 5.5: Accuracies obtained with the DynamoRep and iteration ELA features for the problem classification task of 5$d$ problems. . . . . 106

Figure 5.6: Accuracies obtained with the static ELA features for the problem classification task of 5$d$ problems. . . . . 108
Figure 5.7: Top 20 DynamoRep features with highest feature importance assigned by the RF model for the problem classification task. . . . . 109
Figure 5.8: Accuracies obtained for the algorithm classification task using the DynamoRep and Iteration ELA features, across different experimental settings for 5$d$ problems. . . . . 111
Figure 5.9: Top 20 DynamoRep features with highest median importance for the algorithm classification task, across all four dimensions. . . . . 112

# List of Tables

Table 2.1: Advantages vs. disadvantages of the different problem landscape feature sets. . . . 18
Table 2.2: Advantages vs. disadvantages of algorithm features. . . . 19
Table 2.3: Advantages vs. disadvantages of high-level problem-algorithm interaction features. . . . 22
Table 2.4: Advantages vs. disadvantages of trajectory features. . . . 26
Table 2.5: Summary of works using problem, algorithm, or trajectory-based features and their applications in the domain of continuous SOO. . . . 28

Table 3.1: Parameters of the 12 PSO configurations. . . . 37
Table 3.2: Example of the calculation of the algorithm performance metric for one problem instance. . . . 38
Table 3.3: An example of calculation of the misranking score and the loss for two problem instances. . . . 39
Table 3.4: Mean and median difference between the losses achieved by the ELA model and the transformer model, aggregated across all budgets. . . . 42

Table 4.1: An example of calculation of the pairwise ranking score and the loss for two problem instances. . . . 55
Table 4.2: Parameters of the DE configurations. . . . 71
Table 4.3: Parameters of the PSO configurations. . . . 71

# Abbreviations

AS ... Algorithm Selection
BBOB ... Black Box Optimization Benchmarking
CEC ... Congress on Evolutionary Computation
CMA-ES ... Covariance Matrix Adaptation - Evolution Strategy
CNN ... Convolutional Neural Network
DE ... Differential Evolution
ELA ... Exploratory Landscape Analysis
ES ... Evolutionary Strategy
GA ... Genetic Algorithm
LIO ... Leave Instance Out
LONs ... Local Optima Networks
LPO ... Leave Problem Out
ML ... Machine Learning
PCA ... Principal Component Analysis
PSO ... Particle Swarm Optimization
RF ... Random Forest
SOO ... Single Objective Optimization
STN ... Search Trajectory Network

# Chapter 1

# Introduction

Optimization is a cornerstone of decision-making and scientific inquiry, underpinning a vast array of real-world applications ranging from engineering design and logistics to machine learning (ML) and financial modeling. Among the diverse branches of optimization, single-objective optimization (SOO) focuses on the identification of solutions that minimize or maximize a single criterion.

A continuous SOO problem can be formally defined as the task of finding

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}), \tag{1.1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ is a $d$-dimensional vector of decision variables, $\mathcal{X} \subseteq \mathbb{R}^d$ is the search space, and $f : \mathcal{X} \to \mathbb{R}$ is the objective function (also called the fitness function). The objective function induces an objective landscape, which refers to the mapping from points in $\mathcal{X}$ to their corresponding objective values. A problem instance is fully specified by the tuple $(f, \mathcal{X}, d)$.

An optimization algorithm $A$ takes as input a problem instance and produces a solution $\hat{\mathbf{x}}$ and/or the corresponding objective value $f(\hat{\mathbf{x}})$ after some number of function evaluations, subject to computational or budget constraints. In addition to the final solution, algorithms generate optimization trajectories, such as the sequence of evaluated points $\{\mathbf{x}^{(t)}\}_{t=1}^{T}$, objective values $\{f(\mathbf{x}^{(t)})\}_{t=1}^{T}$, and auxiliary internal states, which can be used for analysis or as inputs to downstream tasks.

Despite the maturity of this field, the diversity and complexity of problem landscapes present persistent challenges. Notably, no single algorithm consistently performs well across all problems. The performance of any optimization algorithm is inherently problem-dependent, and an algorithm that excels on one class of problems may perform poorly on others. This variation in algorithm performance stems from the fact that optimization algorithms may exhibit different behaviors when confronted with functions having distinct properties such as noise levels, multi-modality (presence of multiple local optima that make it challenging to reach the global optimum), separability (the extent to which variables can be optimized independently or require joint consideration due to interdependencies), and search space homogeneity (the degree to which the difficulty of the landscape remains consistent across regions or varies between smooth and rugged areas). Consequently, in practice, selecting an appropriate optimization algorithm for a given optimization problem becomes critical. The traditional approach of trial-and-error or relying on domain expertise for selecting an algorithm is increasingly inadequate in modern, data-rich environments characterized by computational constraints and high-dimensional search spaces.

Aiming to capitalize on the varied performance of different algorithms across sets of different problem instances, the algorithm selection (AS) task [1] targets the automated

identification of a preferred optimization algorithm to solve a particular problem instance. Conventionally, AS is performed by taking into account the properties of the problem instance, which are typically described in the form of a numerical vector representation, also referred to as problem landscape features [2]. Once a problem instance is represented in a vector form, ML models can be used to capture the relation between problem landscape features and algorithm performance, and further identify the best algorithm to solve a given problem instance.

Landscape analysis is a line of research whose aim is to extract features that describe the properties of the problem instances [3]. We typically distinguish between two levels of landscape features. The first type, high-level properties of the fitness landscape (e.g., multimodality, separability, search space homogeneity) [4] and low-level features (calculated using different statistical methods from a sample of candidate solutions sampled from the decision space for a given problem instance using some sampling technique).

In the field of continuous SOO, the most common choice of low-level problem landscape features used to represent problem instances is the Exploratory Landscape Analysis (ELA) features [5]. ELA features are useful for characterizing optimization problems, but they also have some limitations, such as being sensitive to the sample size [6] and the sampling method used to sample solutions from the decision space and not being invariant to transformations such as scaling (multiplying the objective function values or decision variables by a constant factor) and shifting (adding a constant to the objective function values or decision variables) of the problem [7]. Furthermore, AS models using ELA features have shown limited generalizability to unseen problems [8]–[10].

Because of this, the first problem addressed in this doctoral dissertation is the need for novel features for optimization problems. In the past few years, several works have proposed the development of alternative problem landscape features using deep learning approaches [11], [12] and topological data analysis [13], [14]. However, there is a lack of a comprehensive analysis of the complementarity of these features and the degree to which they improve the performance of ELA-based AS models. Transformer models [15], which have achieved remarkable success in domains such as natural language processing [16] and computer vision [17], by effectively capturing complex, high-dimensional dependencies through self-attention mechanisms, could also be explored as a promising direction for extracting informative features of optimization problems.

Apart from the features used by the AS model, another issue that warrants further examination is the model evaluation procedure. In the field of continuous SOO, AS is primarily conducted using the established Black-Box Optimization Benchmarking (BBOB) [18] suite, containing 24 classes of optimization problems. Different instances can be generated from each problem class by applying transformations such as scaling or shifting to the original problems. Such a generation process makes the problem instances within the same problem class highly similar to each other. When using the BBOB benchmark for AS and performance prediction, there are substantial differences in how model evaluation is performed. The two commonly used approaches are referred to as "leave-instance-out" (LIO) and "leave-problem-out" (LPO) [19]. LIO is the easiest evaluation strategy where instances from all problem classes are present in the training set, but only a few instances from each class are used for testing. This means the AS model encounters functions with similar problem landscape features during training and testing. On the other hand, the LPO validation strategy involves testing the model on problem classes that were not present in the training data. Due to the diversity of problems in the BBOB benchmark, the LPO evaluation strategy is much more challenging, since the test set may contain problem instances that are not similar to any of the instances seen during training [9].

Due to such discrepancies in the difficulty of the evaluation, choosing which strategy to

use can substantially influence the results. Other aspects of the experimental setup which vary largely across studies can also lead to experiments which are not directly comparable, such as the algorithm evaluation strategy (fixed-target vs fixed-budget), the choice of the algorithms in the algorithm portfolio, the sample size used to calculate the features, the choice of the ML setup and model, etc. Here, a budget refers to the maximum number of function evaluations or computational resources allocated to an algorithm when solving a problem instance. A fixed-budget evaluation compares algorithms under equal resource constraints, while a fixed-target evaluation compares them based on how quickly they achieve a predefined performance threshold.

Because of this, one goal of this doctoral dissertation is to conduct a comprehensive study including the recently proposed feature groups and the ELA features across evaluation settings of varying difficulty, in order to assess the degree to which they improve the generalizability of ELA-based AS models. Therefore, the second problem addressed in this doctoral dissertation is the need for a comprehensive and fair comparison of existing problem landscape features for the AS task, in evaluation settings of varying difficulty, to provide a deeper understanding of their generalization ability. Additionally, since the AS models have rarely been evaluated outside of the BBOB benchmark, this dissertation strives to use recently proposed strategies for generating new problem instances, either via random or controlled function generators [20], [21], or through combining the existing BBOB problem instances [22], [23].

Additionally, most of the problem landscape features proposed thus far are based on a static sample of candidate solutions from the optimization problem, which is often generated using an artificial sampling strategy such as random, Sobol or Latin hypercube sampling [24]. Therefore, problem landscape features embody characteristics of the entire problem space, regardless of the sections of the problem space that have been encountered by the algorithm in question during its search procedure. It has been demonstrated that the characteristics of the problem space that an algorithm encounters during optimization can substantially differ from the global landscape of the problem being solved [25]. Therefore, features derived from the trajectory of an optimization algorithm during its execution on an optimization problem may provide additional complementary information which can further be used for *i)* dynamic AS, *ii)* analyzing the behavior of optimization algorithms, *iii)* selecting algorithms with complementary behavior to be included in an algorithm portfolio used by an AS model, and *iv)* analyzing the novelty of different algorithms by emphasizing their similarities and differences. This highlights the need for a methodology for identifying features that capture the interaction between an optimization algorithm and an optimization problem.

The purpose of this dissertation is to improve methodologies for feature construction and learning for the domain of continuous SOO, as well as evaluate their utility for the AS task. This includes *i)* the proposal, development, and evaluation of novel feature construction methodologies for SOO problems and algorithms, and *ii)* a comprehensive review, evaluation, and comparison of the existing methodologies.

## 1.1 Research Goals

The goals of this dissertation are:

**G1:** *Introduce and develop new landscape features that effectively capture the characteristics of single-objective continuous optimization problems.*

**G2:** *Conduct a reliable benchmarking study - comparing both the newly proposed landscape*

*features and the state-of-the-art approaches - to assess their ability to generalize automated algorithm selection.*

**G3:** *Design innovative features that model the dynamic interactions (i.e., search trajectories) between single-objective optimization problems and algorithms, and evaluate their utility within automated algorithm selection frameworks.*

## 1.2 Hypotheses

This section addresses the open questions identified in the preceding section and introduces the investigated hypotheses. The initial part focuses on investigating innovative transformer-based methodologies for feature construction within the domain of continuous SOO. The hypothesis related to this research is the following:

**H1:** *Landscape features learned directly via transformer models - operating on raw sample points from a single-objective continuous optimization problem - can accurately predict algorithm performance without any intermediary feature calculation step.*

To further evaluate the proposed transformer-based features and compare them to existing ELA features, we conduct a study to evaluate the generalizability of AS models across benchmark suites. This study addresses the following hypothesis:

**H2:** *The ability of an automated algorithm selection strategy to generalize across distinct benchmark suites depends critically on (a) the extent of problem coverage provided by the suite and (b) the underlying performance distribution of the algorithms evaluated on it.*

Furthermore, we provide a comprehensive evaluation of the proposed transformer-based features, existing ELA features, and recently developed features based on deep learning and topological landscape analysis. The goal of this study is to test the following hypothesis:

**H3:** *Novel-developed landscape features and their combinations outperform traditional landscape features in automated algorithm selection.*

Apart from problem landscape features, we also investigate the applicability of features calculated from the trajectory of the optimization algorithm produced during the search process, which is addressed in the last hypothesis:

**H4:** *Simple, low-cost features extracted online from the search trajectory of an optimization algorithm can serve as effective inputs for automated algorithm selection, yielding performance on par with - or superior to - existing static feature sets.*

## 1.3 Scientific Contributions

This dissertation contributes to the field of automated AS for continuous SOO through three core scientific contributions, each addressing a major gap identified in current literature. The contributions correspond to the research goals G1-G3 and align with the hypotheses H1-H4 introduced earlier.

**C1:** *Transformer-based feature construction from raw problem samples.* The first contribution of this work is the introduction and evaluation of transformer-based models for constructing representations of optimization problems directly from raw sample points. The proposed methodology leverages transformer architectures, which use self-attention mechanisms to identify and encode complex patterns in raw data. This allows for the automated learning of high-dimensional, expressive features that aim to capture the structural characteristics of problem landscapes.

**C2:** *Systematic benchmarking study on feature generalizability.* The second major contribution is a large-scale empirical study that rigorously benchmarks the performance of

various feature groups-including ELA features, deep learning-based features, and topological landscape analysis features. The study evaluates how well these features support generalization under evaluation strategies of increasing difficulty. This contribution establishes new baselines, exposes limitations in existing methodologies, and highlights the importance of evaluation rigor in AS research.

**C3:** *Introduction of dynamic, trajectory-based features for modeling algorithm–problem interactions.* Recognizing the limitations of static, sample-based features, the third contribution investigates the use of trajectory-based features-i.e., features derived from the behavior of an algorithm during its search process. These dynamic features are computed from the search trajectory and are designed to capture algorithm-specific behavior in response to a given problem landscape. Unlike global features that describe the entire search space, trajectory features emphasize the local and adaptive interaction between algorithm and problem. The dissertation proposes a set of lightweight, low-cost trajectory descriptors and explores their application in multiple contexts: improving AS model input, understanding algorithm complementarity, and detecting behavioral similarity among algorithms.

## 1.4 Thesis Structure

This dissertation is organized as follows. Chapter 2 reviews relevant background literature on feature representation techniques for continuous SOO. Chapter 3 explores transformer-based learning of landscape representations from raw samples of the optimization problem. Chapter 4 focuses on evaluating the generalizability of AS models using the existing and newly proposed transformer representations. Chapter 5 explores dynamic features extracted from search trajectories and their problem-algorithm interaction modeling potential. Chapter 6 concludes the dissertation and outlines directions for future work.

# Chapter 2

# Background and related work

In this chapter, we first present the typical way of approaching an AS problem, and describe the components of an AS pipeline. We then give an overview of the feature representations used in continuous SOO. Specifically, we introduce *i)* problem landscape features which are used to characterize an optimization problem, *ii)* algorithm features used to characterize an optimization algorithm and *iii)* trajectory features which characterize the interaction between an optimization problem and algorithm. In this case, a trajectory refers to the ordered sequence of solutions that an optimization algorithm generates during its search process, together with their corresponding objective (fitness) values. It represents the path the algorithm takes through the decision space (or equivalently, the fitness landscape) as it iteratively evaluates, updates, and moves from one candidate solution to the next.

## 2.1 Machine Learning Pipeline using Optimization Features

Figure 2.1 illustrates an ML pipeline that includes essential components for calculating or learning features related to problem landscapes, algorithm characteristics, and problem-algorithm interactions. It also highlights several learning tasks - such as AS, problem classification, and complementarity analysis of benchmark problem suites - where these features serve as inputs to the ML model. It is important to note that the specific features and components involved may vary depending on the particular ML task being solved.

The main components of such a pipeline are:

- **Problem Portfolio** - A set of optimization problem instances for which we want to find an optimal algorithm instance, analyze their coverage, or predict their high-level properties (such as multimodality, funnel and global structure). In the continuous SOO domain, examples of such functions are the sphere function, ellipsoidal functions, discus, etc. These problem instances are structured in problem benchmark suites, the most commonly used ones being the Black Box Optimization Benchmarking (BBOB) [18] suite and the IEEE Congress on Evolutionary Computation (CEC) Special Sessions and Competitions on Real-Parameter Single-Objective Optimization [26]–[29] suites. For a sound (academic) evaluation of the ML models, the problem portfolio should cover problem instances that are diverse, challenging, and discriminating of algorithms' performances. In recent years, there has been a growing interest in the optimization field regarding the generation of new optimization problem instances in an automatic manner [21], [30]–[33]. Generating novel problem instances has the potential to introduce previously unexplored optimization challenges in ongoing research.

- **Algorithm Portfolio** - A set of optimization algorithm instances that are candidates

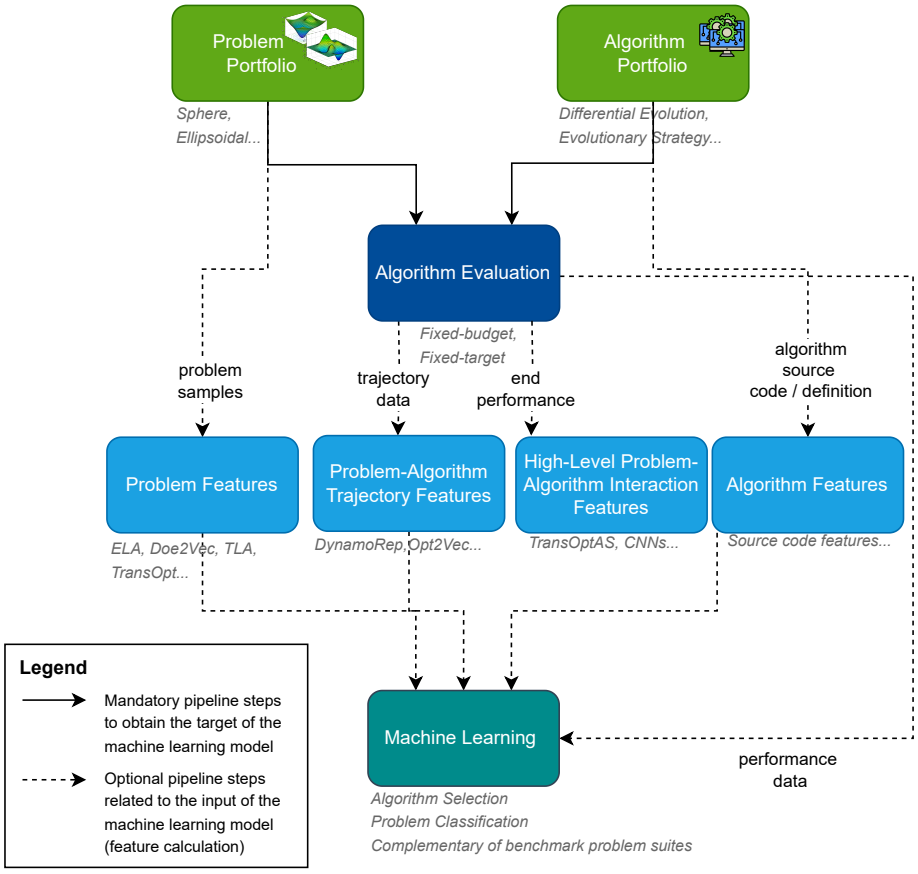



Figure 2.1: A typical machine learning pipeline using optimization features.

for solving an optimization problem instance. Some examples in continuous SOO are Differential Evolution [34], Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [35], and Particle Swarm Optimization (PSO) [36] algorithms. In addition, modular optimization frameworks are also utilized such as ModCMA [37], [38] and ModDE [39].

- **Algorithm Evaluation** - The process of running the optimization algorithm instances on the optimization problem instances and recording their performance in terms of some evaluation/performance metric. Two standard evaluation scenarios include the fixed-budget and fixed-target evaluations. In the **fixed-budget scenario,** the algorithm instance is allowed to use a predefined budget and one studies the solution quality or, if the value of the global optimum is known, the *target precision,* defined as the difference between the quality of the solution recommended by the algorithm (typically, but not necessarily, the best evaluated one) and that of the global optimum. In the **fixed-target scenario,** the algorithm is tasked with finding a solution of a certain quality, and we study the budget that is needed by the algorithms to find a solution of at least this quality. We recall that, in black-box optimization, the *budget* is often specified in terms of function evaluations or iterations, not in terms of (CPU or wall-clock) time. A third commonly used set of performance measure considers the *anytime performance* of the algorithms, measured through empirical cumulative distribution functions [40] or directly through empirical attainment functions [41].

- **Problem Landscape Features** - Problem landscape features are numerical representations of problem instances based on a sample from the entire problem instance. They are typically calculated *before* the optimization algorithm is run, meaning that additional function evaluations are required for the feature construction process. To calculate them, a set of candidate solutions is sampled from the decision space (using methods such as random sampling or Latin hypercube sampling [24]) and their objective values are analyzed to derive descriptive statistics and structural indicators. The characteristics considered include the dispersion of fitness values (capturing global structure), autocorrelation length (measuring ruggedness), skewness and kurtosis of the fitness distribution (indicating asymmetry or heavy tails), and correlation between variables and objective values (providing insight into separability). We provide an overview of problem landscape feature construction in Section 2.3.

- **Algorithm Features** - These features characterize the algorithm instance. They are generated by analyzing the source code [42] or derived from algorithm parameters that remain constant and do not depend on the problem being solved. It is the low resourced investigated feature group. An overview of algorithm features is given in Section 2.4.

- **High-level Problem-Algorithm Interaction Features** - These features rely on high-level information about the optimization problems on which the algorithms are executed, rather than capturing the trajectory of the optimization runs. In essence, these features connect high-level information about a set of optimization problems with the final performance achieved by an algorithm. Different approaches exist focusing on features based on their performance [43], features based on graph embeddings [44] and graph neural networks (GNNs) [45], and features based on configuration settings for modular algorithm frameworks [46], [47]. Section 2.5 provides an overview of the algorithm features.

- **Trajectory-based Features for Capturing Problem-Algorithm Interaction** - As opposed to problem landscape features, which describe the whole landscape of a problem instance, without taking care about the landscape visited/observed when an algorithm instance is run on it, trajectory-based features involve the usage of samples generated by optimization algorithms during their execution on a problem instance. This type of representation captures interactions between the problem instance and the optimization algorithm. An overview of trajectory-based feature constructions for capturing problem-algorithm interaction is given in Section 2.6.

- **Machine Learning** - The problem landscape features, algorithm features, trajectory-based features, or a combination of different groups can be used as input data for an ML model in different learning tasks. The three most researched tasks are presented below:

  - AS - One approach for training an AS model is to treat it as a *multi-class classification problem* [8], with the objective of identifying a single optimal algorithm for the specific problem instances. Alternatively, it can be treated as a *multi-label classification problem,* where multiple algorithms may be selected (e.g., two or three different algorithms with similar performance). Other learning scenarios include solving a *regression problem* [48]–[51], in which the ML model predicts the performance achieved by each algorithm, or *ranking* [9], where the aim is to rank the algorithms according to a specific evaluation metric. A comprehensive study comparing approaches for AS in continuous SOO - through different learning tasks (multi-class classification, pairwise classification, regression) and performed with different ML algorithms (i.e., Random Forest [52], XGBoost [53], TabPFN [54], and FT-Transformer [55]) - is available in [56].
  - Problem Classification - In this task, features serve as input to a model to predict either the problem represented (for problem landscape features) or the problem being solved (for problem-algorithm trajectory features) [57], [58]. Identifying this information can assist users and researchers in applying their domain knowledge to enhance the optimization process.
  - Complementarity of benchmark problem suites - In this task, various sets of problem instances are represented by problem landscape features, and their diversity is examined using unsupervised learning techniques like clustering [59], dimensionality reduction [60], and graph algorithms [61].

## 2.2 Problem Portfolios

Here, an overview of the existing benchmark suites for numerical single-objective optimization, together with problem generators that exist to generate new problem instances is presented.

### 2.2.1 Benchmark suites

In the field of single-objective continuous optimization, algorithm instance performance is commonly assessed experimentally, for example on the benchmarks provided by the IEEE Congress on Evolutionary Computation (CEC) Special Sessions and Competitions on Real-Parameter Single-Objective Optimization [26]–[29] and the Genetic and Evolutionary Computation Conference Black-box Optimization Benchmarking (BBOB) [18] workshops.

The CEC2013 benchmark suite contains 28 problems, CEC2014 contains 30 problems, CEC2015 contains 15 problems, and CEC2017 contains 29 problems. These problem suites

include unimodal functions (with a single global optimum, useful for testing convergence speed), multimodal functions (with many local optima, testing global search ability), hybrid functions (constructed by combining multiple basic functions, creating irregular and heterogeneous landscapes), and composition functions (blending different regions of the search space with distinct properties, to simulate real-world complexity). The BBOB benchmark suite defines 24 base problems, from which different instances can be generated by applying transformations, such as shifting and scaling. The BBOB problems are grouped into categories reflecting their structural properties. These categories typically include separable functions (where variables can be optimized independently), functions with low or moderate conditioning, unimodal functions with high conditioning, multimodal functions with many local optima, and functions with weak global structure. Each category is designed to represent a distinct type of challenge that optimization algorithms commonly face. While the problem classes are intentionally diverse, providing a wide coverage of landscape characteristics, the instances generated from the same problem class are structurally similar to each other.

Other benchmark suites that can are also utilized in this domain are AI-Roomi repository [62] and Nevergard [63]. AI-Roomi repository is a benchmark suite of unconstrained single-objective benchmark problem instances, while Nevergard developed by Meta consists of several benchmark suites from which yabbob, yasmallbbob, yatuningbbob, yaparabbob, and yaboundedbbob (all derived from the BBOB) can be used for single-objective continuous optimization.

### 2.2.2 Problem generators

It has been shown that the CEC and the BBOB benchmark suites provide poor coverage of the space of continuous optimization problems [30], [60], [64]. In recent years, there has been a growing interest in the optimization field regarding the generation of new optimization problem instances in an automatic manner [22], [33]. Generating novel problem instances has the potential to introduce previously unexplored benchmarks in ongoing research. This capacity to generate problem instances can also be leveraged to address intricate real-world optimization challenges. By producing problem instances that closely resemble the characteristics of the target problem, we can establish a rigorous test set for algorithm evaluation and configuration.

In [30], a method for generating diverse and challenging benchmark problem instances in continuous black-box optimization has been proposed. Each problem instance is represented by a feature vector representing its problem landscape characteristics. By mapping the problem landscape features onto a two-dimensional instance space, it becomes possible to visualize the positions of current benchmark problem instances and discern their similarities and distinctions. Further, through genetic programming, functions with controllable problem landscape features are evolved, guided by convergence toward selected target points (i.e., problem instances) in the instance space, ensuring comprehensive coverage of the space. Recently, a study [65] has shown that employing a genetic programming approach guided by problem landscape features is dependent on the selection of the features and may not consistently yield satisfactory problem instances. In addition, the choice of a distance measure between the problem instances should be smartly done to evolve functions that adequately reflect the characteristics of the target landscape.

In [21], a generator of artificial problems was presented, aimed at creating problem instances of varying complexity in the shape of tree representations, which were employed as training data for a recommendation model for selecting the most suitable algorithm instance. This generator constructs tree representations by randomly choosing mathematical operands and operators from a predefined set, each with its assigned selection probabil-

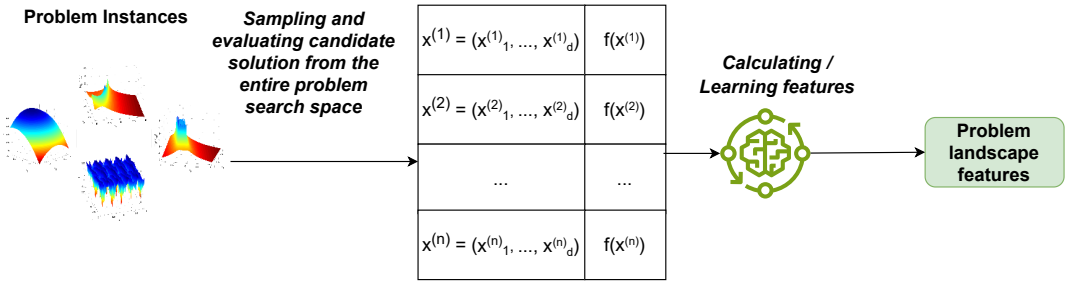



Figure 2.2: The general pipeline of calculating problem landscape features.

ities. This approach enables the swift creation of a large number of problem instances. To augment the variety of generated problem instances, encompassing elements like noise, multi-modal landscapes, and intricate variable relationships, an operation called "difficulty injection" was introduced to modify the tree representation. Furthermore, the study also contemplated a “tree-cleaning" operation to simplify the tree representation by eliminating redundant operators.

Recently, an approach has been proposed where a novel problem instance is created by blending two chosen BBOB problem instances through a convex combination, where a weighting factor regulates the interpolation [22]. Furthermore, this research has been extended to encompass affine combinations of multiple problem instances (beyond just two problem instances) [33]. They cover a larger part of the problem landscape space and allow a fine-grained analysis of algorithm performance on problems with different characteristics.

Recently a new study has also explored the GKLS [31] generator in a scenario of black-box optimization and compared its coverage with the BBOB and CEC benchmark suites [66], which is a popular benchmark suite in deterministic optimization. It generates diverse problem instances for multi-modal, multi-dimensional box-constrained optimization. It offers eight primary problem classes, two for each of dimensions 2, 3, 4, and 5, totaling 100 commonly used problem instances. It can also generate various new problem types based on input parameters.

A recent approach for problem instance generators is predominantly focused on creating a point cloud that highlights the intended ELA features [32], leading to the desired characteristics of the fitness landscape. This point cloud is utilized as the training dataset for a neural network, which functions as a surrogate model to generate novel benchmark functions. The method not only enables interpolation within the ELA feature space of benchmark functions but also extends the capability for extrapolation beyond the convex hull of this region. Since it is a new problem instance generator, it has not been utilized in algorithm selection studies yet.

## 2.3 Problem Landscape Features

In this section, we describe approaches that calculate problem landscape features using a set of candidate solutions, sampled from the decision space of the problem instance via some non-adaptive or via a deterministic sampling technique. What characterizes these features is that they aim to represent the problem instance, no matter which algorithm instance is further run. Figure 2.2 presents the general pipeline of calculating problem landscape features.

Several approaches for capturing characteristics/properties of continuous SOO prob-

lems have already been proposed. They can be broadly categorized as providing high-level and low-level features.

High-level fitness landscape analysis captures optimization problem characteristics such as the degree of global structure (none, low, medium, high), separability, variable scaling, multi-modality, search space homogeneity, etc. For example, the features related to ruggedness, neutrality, gradients, global landscape structure, deception, and searchability have been explored together with their correlations with the diversity rate-of-change metrics of the behavior of the particle swarm algorithm [67]. The results have shown links between particular features and PSO behavior. In contrast to some of the low-level landscape features, these features are designed to be highly interpretable. The rest of this section describes existing low-level features, or more precisely, *Exploratory Landscape Analysis* [5], *Topological Landscape Analysis* [14], and *deep learning-based approaches* [12].

### 2.3.1 Exploratory landscape analysis

Exploratory landscape analysis (ELA) [5] is an approach to characterize black-box optimization problem instances via numerical measures that each describe a different aspect of the problem instances. ELA features can be further distinguished into *cheap* features that are computed from a fixed set of samples and *expensive* features that require additional sampling (e.g., they may require running some local search variants or other sequential sampling approaches) [68]. Cheap features are preferable when the evaluation budget is limited or when features must be computed for a large number of problem instances quickly, as they rely only on a fixed set of pre-sampled solutions and do not require further evaluations. This makes them well-suited for scenarios such as algorithm selection in online settings, real-time decision-making, or large-scale benchmarking, where computational efficiency is crucial. Expensive features may be more suitable in offline settings, small to medium-scale studies or for situations where problem evaluations are cheap relative to algorithm runs, while cheap features are favored when evaluation costs are high or scalability is the primary concern. A convenient way to compute ELA features is provided by the *flacco* R package [69]. This package offers 343 different feature values split into 17 feature groups, including *dispersion*, *information content*, *meta-model*, *nearest better clustering*, and *principal component analysis*. Recently, a Python version of the package, offering some additional feature sets from local optima networks, has been published [70].

The original ELA features [5] were grouped into six categories:

- The **convexity** of an optimization problem is captured by observing the difference in the objective value of a point which is a linear combination of two randomly sampled points and the convex combination of the objective values of the two sampled points.

- The **distribution of the objective function values** is characterized by its skewness, kurtosis, and the number of peaks based on a kernel density estimation of the initial design's objective values.

- **Local search** features are extracted by running a local search algorithm and hierarchically clustering the considered solutions in order to approximate problem properties. For instance, the number of clusters is an indicator of multi-modality, while the cluster sizes are related to the basin sizes around the local optima.

- The **levelset** features split samples into two classes based on whether the value of the objective function falls above or below a certain threshold. Linear, quadratic, and mixture discriminant analysis is used to predict whether the objective values fall

below or exceed the calculated threshold. The intuition behind this is that multimodal functions should result in several unconnected sublevel sets for the quantile of lower values, which can only be modeled by the mixture discriminant analysis method. The extracted low-level features are based on the distribution of the resulting misclassification errors of each classifier.

- The construction of **meta-model** features requires the use of linear and quadratic regression models, with and without interaction effects, which are fitted to the sampled data. The features are then calculated by taking the adjusted coefficient of determination R2 of the models, indicating model accuracy, as well as the model coefficients. Such features are useful for extracting function characteristics, since functions with variable scaling will not allow a good fit of regression models without interaction effects, and simple unimodal functions could be approximated by a quadratic model.
- **Curvature** features estimate the gradient and Hessians from samples of the function and use their magnitudes to quantify the curvature.

Cell mapping features [71] partition the problem landscape into a finite number of cells:

- **Angle** features are based on the location of the worst and best element within each cell since opposite locations of these elements indicate a trend within a cell. More precisely, the features observe the mean and standard deviation of distances from the cell center to the best / worst element, and of the angle between these three points.
- **Convexity** features represent each cell by the element closest to the cell center and compare the objective function values of neighboring cells to capture convexity/concavity.
- **Gradient homogeneity** features compute the length of the sum of the gradients between each element and its nearest neighbor within the cell.
- **General cell mapping** features interpret the problem landscape cells as absorbing Markov chains. Cells that are visited infinitely often, once they have been visited once, are referred to as periodic cells, while all other cells are transient cells. Different types of features are then extracted based on the probability of moving from one cell to another, the number of cells of different types that appear, the size of the basins of attraction, etc.

Over time, additional ELA features have also been introduced:

- **Basic** features include minimum and maximum of the objective function, number of samples, etc. based on an initial sample of points.
- **Dispersion** features [72] compare the dispersion among all the samples and among a subset of these points.
- **Linear Model** features [69] first build several linear models with the objective variable being the dependent variable and all the remaining variables being treated as explaining variables. Based on the obtained model coefficients, statistics are calculated that are used as features.
- **Nearest Better Clustering** features [73] are computed by comparing samples to their closest neighbor and to their better closest neighbor.

- Features based on **Principal Component Analysis** [69] describe the relative amount of principal components required to explain a predetermined amount of variability.
- **Barrier Tree** features [74] describe the landscape of an optimization problem using trees where saddle points are represented with regular nodes while minimums are described by leaves. Features are then generated by computing tree descriptors such as the number of levels and the distance between the leaves.
- **Information Content** features [75] try to capture information content of the landscape such as smoothness, and ruggedness. This is achieved by taking into account the distance between neighbours and changes in fitness values between them.

The ELA features are accessible through the software packages *flacco* [69] and *pflacco* [70]. However, it is not a comprehensive implementation of all existing features. For instance, information epistasis [76] or auto-correlation [77], initially designed for discrete variables, can be easily adapted to continuous ones. In addition, the same is true for the use of entropy [78] as a measure.

ELA features are the most commonly used problem instance landscape features, however they come with a lot of limitations: their poor robustness to sampling strategy and sample size [6], [79], [80], and invariance to transformation in the problem space (i.e., shifting, scaling, or rotation of a problem instance) [7], [60], [81]. Recently, a new study highlights how many of these features are sensitive to absolute objective values (i.e., lack of shift and scale invariance), causing bias in automated AS models and hindering their ability to generalize to new problem instances [82]. To mitigate this, it has been proposed to normalize the sampled data before computing ELA features.

ELA features have also been used to characterize Neural Architecture Search [83] landscapes across three typical image classification datasets: MNIST, Fashion, and CIFAR-10 [84]. This analysis reveals distinct characteristics of the Neural Architecture Search landscapes of these three datasets. ELA features have also been used to perform complementarity analysis of the hyperparameters optimization landscapes (HPO) and classical BBOB problems from [18], for black-box optimization [85] and for XGBoost learner [86]. Additionally, the ELA features have been used to identify cheap representative functions of 20 automotive crashworthiness problems which are expensive to evaluate [87], where it was also shown that these problems exhibit landscape features different from the classical BBOB benchmark suite.

### 2.3.2 Topological landscape analysis

Topological Landscape Analysis [14] features characterize optimization problem instances based on Topological Data Analysis, which is an approach to analyzing and obtaining features from a finite set of data points, also referred to as a *point cloud*. Similarly to the ELA features, the point cloud is constructed based on a set of samples from the optimization problem. Once the samples of the objective function are collected, they are sorted by objective value and divided into multiple layers. Each layer's point cloud reveals the structural characteristics specific to that layer. For each point cloud, a persistence diagram is generated, capturing the topological features within the layer. These persistence diagrams are then transformed into finite-dimensional vector representations known as *persistence images*. Persistence images enable each layer's samples to be represented as a feature vector, where pixel values from the image provide a meaningful encapsulation of the underlying structure in vector form.

The goal of Topological Data Analysis is to discover topological structures such as spheres, torus, connected components, or even more complicated surfaces and manifolds,

which persist across scales, independently of certain transformations (e.g. rotations, scaling). Discovering such structures can help characterize the point cloud using features that capture the existence of different structures across different scales.

Apart from their original version [14], an improved version called TinyTLA requiring smaller sample sizes for feature calculation has also been proposed in [13]. In particular, as opposed to the original TLA features which obtained classification accuracies ranging from 60% to 80% in dimensions $d \in \{2, 3, 5\}$ with $1000d$ samples, the TinyTLA approach can already reach similar accuracies with 400 samples ($80d$ to $200d$ sampels).

Here, instead of splitting the problem samples into layers, the information from the entire set of problem samples is captured using a matrix of distances between the x-values and y-values of each pair of samples. The thereby obtained distance matrix is used to further compute the persistence diagrams, persistence images, and feature representations as in the original TLA. The TinyTLA features have also been tested on the BBOB problem classification task with an extensive analysis of parameter influence and an initial evaluation for the AS task on the BBOB benchmark.

### 2.3.3 Deep learning-based approaches

Here, more details about low-level features learned by different types of deep neural networks will be provided. Most of these features have been developed in parallel with the work presented in this thesis.

#### 2.3.3.1 Features learned by using convolutional neural networks

A set of low-level features learned by convolutional neural networks (CNNs) [88] are presented in [89], [90]. To calculate them, first, for each problem instance involved in the training data (i.e., BBOB problem instances), a set of candidate solutions is generated using a Latin Hypercube sampling (other sampling techniques can also be used). The set of candidate solutions is further used to calculate a fitness map, which is a $2d$-image with a single channel in a [0,1] range. To generate the image, the objective solution values are normalized in the range of [0,1], and each candidate solution is mapped into a Cartesian plane at the location defined by the decision variables and the objective solution value defines the color range (gray-scaled) of each pixel. One weakness of this transformation is the information loss since two different candidate solutions that are close will be mapped to the same pixel. Transforming the set of candidate solutions into a $2d$ image allows utilizing a CNN to compute low-level features for the problem instances. In [89], the CNN that has been utilized is the ShuffleNet v2 [91] due to the competitive performance achieved in object classification in computer vision. The study involves the 24 BBOB problem classes and 124 instances per problem in $d = 2$. The model has been trained in a supervised manner in a task of AS, where for each problem instance the most appropriate modular CMA-ES out of an algorithm portfolio of 32 configurations is selected. The latent representations that are obtained by the CNN can be used as low-level features. In [90], this work has been extended for high-dimensional data rather than two or three dimensions. For this purpose, four dimensionality reduction techniques have been investigated to reduce the fitness map dimensionality into two dimensions that are further utilized by the 2d CNN to learn the low-level features. Here, 24 BBOB problem classes are used with 150 problem instances per class. The training has been performed in a supervised manner in a problem classification task where the problem instances are classified into one of the three classes based on the high-level properties i.e., multimodality, global structure, and funnel structure. We need to point out here that the low-level features learned for the same problem instances across both studies are completely different as a fact of the different supervised learning scenarios

(i.e., AS vs. multi-class problem classification).

#### 2.3.3.2 Features learned using a point cloud transformer

When calculating the fitness map for high-dimensional problem instances ($d > 2$) there is a loss of information from high-dimensional to low-dimensional space. To mitigate this, point cloud transformers have also been utilized to calculate low-level features. In [90], the authors modified the original work on point cloud transformers that use the idea of convolutions that operate on the edges within a $kNN$-graph [92]. Here, instead of convolutions on the edges, the embedding operates on the node of the $kNN$-graph. This means that every candidate solution is embedded into its local neighborhood (similar candidate solutions). This has been performed since these features have been learned in a supervised manner in a problem classification task where the problem instances are classified into one of the three classes based on the high-level properties i.e., multimodality, global structure, and funnel structure.

#### 2.3.3.3 DoE2Vec features learned using a variation autoencoder

DoE2Vec features [11] are low-level features learned in an unsupervised manner by using a variational autoencoder [93]. To learn them, first, a dataset of candidate solutions, initially within the domain of $[0,1]^d$ ($d$ is the dimension of the problem instances) is generated using a sampling technique. Next, the candidate solutions are evaluated using a set of problem instances that are randomly generated using the random function generator from [94], which is a modification of the generator proposed in [21]. This evaluation allows the collection of training data with different complexity that covers a wide range of problem instances. Further, all objective solution values are first re-scaled within the range of [0,1] and are then used as input features to train the variational autoencoder. These features not only demonstrate promise in identifying similar surrogate problem instances that are inexpensive to evaluate but also have the potential to significantly enhance performance when used alongside traditional ELA features in problem instance classification tasks.

#### 2.3.3.4 TransOpt features learned using a transformer

TransOpt features [95] are low-level features learned in a supervised manner by using a transformer architecture [96]. This architecture has been trained in a supervised manner on two different tasks. These features are one of the contributions from this thesis and are explained in detail in Chapter 3.

#### 2.3.3.5 DeepELA features learned using a transformer

DeepELA [12] is a methodology involving the unsupervised training of transformer models to produce representations of optimization problems which are invariant to problem transformations. This is accomplished by training the model to produce the same representation for a problem instance and its augmented variant. The augmentation is accomplished through transformations that do not alter the underlying optimization problem, such as rotations and inversions of decision variables and randomization of the sequence of decision and objective variables.

Four transformers have been pre-trained on large sets of randomly generated optimization problems to grasp intricate representations of continuous single- and multi-objective landscapes. The DeepELA features have been evaluated in the problem classification task classifying the BBOB single-objective problem based on their high-level characteristics defined in the benchmark suite. In addition, they have been evaluated in automated AS

tasks on the BBOB benchmark suite and four different multi-objective benchmark suites. A study evaluating the complementarity of DeepELA and classical ELA features was recently presented in [97], with a following study involving TransOpt features [98].

#### 2.3.3.6 Random filter mappings

As opposed to using trained deep learning models to extract problem landscape features, [99] proposes the extraction of features using randomly initialized filters. The procedure starts with the initialization of a set of filters with random sizes, weights and radii. Each individual filter is then applied at randomly selected anchor points (samples of the objective function). In particular, the filter application involves the multiplication of the filter weights with the pairwise distances of a subset of the problem samples which are in the vicinity of the anchor point (have a distance which is less than the dimension-adjusted radius of the filter). The thereby obtained filter responses are then aggregated to obtain the problem landscape features. These features have been evaluated for the AS task, as well as the task of problem classification of the BBOB problem instances into the 24 problem classes, and recognizing their high-level properties.

### 2.3.4 Advantages vs. disadvantages of problem landscape features

After summarizing the latest trends in problem landscape features, Table 2.1 presents their advantages and disadvantages.

Table 2.1: Advantages vs. disadvantages of the different problem landscape feature sets.

| **Problem features** | **Advantages** | **Disadvantages** |
|---|---|---|
| ELA | - Easy use through R, Python, and GUI<br>- Most commonly-used problem landscape features<br>- Explainable to some degree | - Sensitive to sampling strategy and sample size<br>- Noninvariant to problem shifting, scaling, and rotation<br>- Poor generalization on new unseen instances when plugged into an ML model<br>- Computationally costly |
| TLA | - Invariant to shifting, scaling, and rotation | - Black-box features<br>- Can produce a very high-dimensional feature vector, may require dimensionality reduction before usage |
| Learned by CNNs | - More applicable for 2D and 3D problems<br>- Does not require any feature engineering | - Black-box features<br>- Loss of high-dimensional information by representing the data as a 2D fitness map<br>- Depends on the problem and algorithm portfolios used for learning |
| Learned by point cloud transformer | - Applicable to all problem dimensions, though may require model retraining for higher dimensions | - Black-box features<br>- Depends on the problem portfolios used for learning |
| DoE2Vec | - Does not require any feature engineering | - Black-box features |
| TransOpt | Does not require any feature engineering | - Black-box features |
| DeepELA | - Does not require any feature engineering | - Black-box features |
| Random filter mappings | - Does not require any feature engineering | - Black-box features<br>- May produce irrelevant features |

## 2.4 Algorithm Features

Algorithm representations are focused on calculating features that characterize the algorithm instance. While the development of algorithm features has garnered somewhat lower research interest compared to the development of problem landscape features, in this section we present the features derived directly from the source code.

### 2.4.1 Algorithm features based on source code

The extraction of features from the algorithm source code and from the abstract syntax tree has been proposed in [42]. The features extracted from the source code capture the cyclomatic complexity, maximum indent complexity, number of lines of code, and size in bytes for the entire source code as well as aggregations of these properties across different regions of the code. On the other hand, the features extracted from the abstract syntax tree involve the conversion of the abstract syntax tree obtained during the compilation of a given algorithm into a graph representation and extracting various graph properties such as node count, edge count, transitivity, node degree, clustering coefficient, depth, etc.

### 2.4.2 Advantages vs. disadvantages of algorithm features

Table 2.2 presents their advantages (pros) and disadvantages (cons) of the algorithm features based on the source code.

Table 2.2: Advantages vs. disadvantages of algorithm features.

| **Algorithms features** | **Advantages** | **Disadvantages** |
|---|---|---|
| Based on source code | - May be used to compare different programming implementations of the algorithms and further investigate which one has better performance | - These features are ineffective for automated algorithm configuration or parameter tuning, as parameter differences are typically evident only during execution, not in the code.<br>- Features extracted from the source code depend highly on the programming language and the specific implementation, leading to potential discrepancies even for the same algorithm. |

## 2.5 High-Level Problem-Algorithm Interaction Features

These features are based on general characteristics of the optimization problems where the algorithms are applied, instead of describing how the algorithm's search process unfolds over time. Essentially, they establish a link between the problem's high-level properties and the algorithm's final performance, but they do not capture details of the algorithm's behavior during its run. Figure 2.3 presents the general pipeline of calculating high-level problem-algorithm landscape features.

### 2.5.1 Algorithm features based on performance

Performance2Vec [43] is a methodology for constructing algorithm representations based on the performance they achieve on a set of benchmark problems. Here, the vector representation consists of the algorithm instance performance obtained on each benchmark problem separately, so they can be assumed as different features. Algorithms' performance

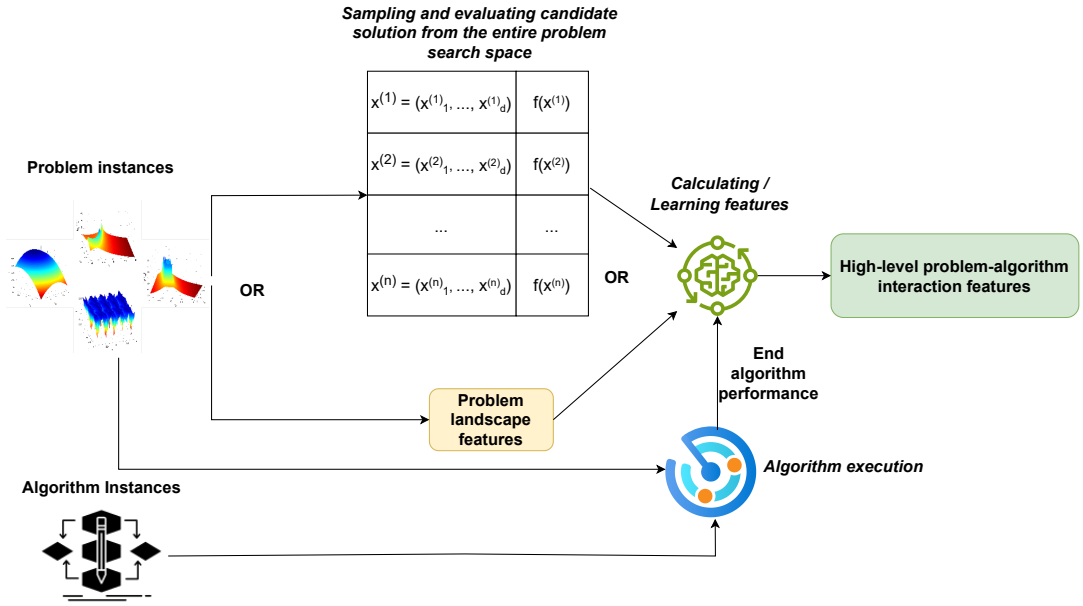



Figure 2.3: The general pipeline of calculating problem landscape features.

can be defined in terms of different metrics, using simple statistics like mean or median, or more complex ranking schemes like Deep Statistical Comparison [100].

### 2.5.2 Algorithm features based on problem landscape features used by performance prediction models

This type of representation is generated by observing the importance assigned to ELA features by explainability methods applied to ML models trained for performance prediction. It involves the use of the SHAP method, which can assign feature importance on a *global* level (i.e., on a set of problem instances) and on a *local* level (i.e., on a particular problem instance). A single-target regression model is trained to predict the performance of the algorithm instance using the training problem instances described by their ELA features. The model is then used for making predictions on the problem instances from the test dataset. Next, by using the SHAP method, the Shapley values that are the contributions of each landscape feature to the performance prediction are calculated. If the input to the model are $p$ landscape features, averaging the Shapley values across all problems for each landscape feature separately gives a $p$-dimensional representation of the algorithm behavior. This type of representation encodes the interactions between the landscape features and the algorithm performance. Such representations have been previously used to find similar algorithm instances behavior on a set of benchmark problem instances [51] and to understand which module from a modular CMA-ES is active based on the performance data [49].

### 2.5.3 Algorithm features using graph embeddings

Algorithm features can also be constructed by leveraging their interactions with different entities from the optimization domain. In [44], the performance prediction task was addressed as a link prediction task, which aims to identify if an algorithm can solve a given problem instance with some specified error. The methodology, in this case, involved the

construction of a knowledge graph, where some of the nodes describe aspects of the problem instances (their corresponding problem class, high-level features, and ELA features), while other nodes are related to algorithm descriptions (their parameters). The algorithm and problem instance nodes are linked if the algorithm can solve the problem instance with some specified error threshold. Such a representation of the algorithm and problem instance properties allows the application of standard knowledge graph embeddings for deriving representations of the problem instances and algorithm instances, which can then be used for performance prediction. We need to highlight here that this approach of learning features can produce either algorithm features if the node is representing an algorithm instance or problem instance features if the node is representing a problem instance. In the case of problem instance features, they differ from the low-level landscape features described in Section 2.3 since they fuse information not only for the landscape but also consider the interaction of an algorithm instance on each problem instance through its end performance.

### 2.5.4 Features based on configuration settings for modular algorithm frameworks

Another approach for extracting features from algorithm configurations involves using the importance or contribution of operators and modules to the performance of the algorithm configuration. Two approaches are presented here:

- Features based on functional ANOVA (fANOVA) - Problem class-specific datasets are initially generated, in which modules are used as features of the modular algorithm variants, with numerous variants incorporating different module values being executed [46]. The corresponding performance on the problem class is treated as the target within each dataset, thereby capturing the relationship between the algorithmic modules and performance in that specific problem class. These datasets are then provided as input to f-ANOVA [101], through which the variance in performance is decomposed and systematically attributed to individual modules and their combinations. When variants are defined by $n$ modules, the analysis quantifies the effect (i.e., importance) on performance for $n$ individual modules, $\binom{n}{2}$ pairwise module interactions, and $\binom{n}{3}$ triple module interactions. Estimation of higher-order interactions is considered computationally expensive. The complete set of effects (singles, pairs, and triplets) is subsequently used as a feature vector, and by comparing these vectors across different problems, problems can be identified in which the modular frameworks exhibit similar module interactions that describe their performance.

- Features based on Shapley analysis - Similar to features generated by fANOVA, Shapley analysis can also be used to estimate the contribution of each module to performance [102]. However, a limitation of this approach is that only the contributions of individual modules are calculated, as estimating higher-order contributions is computationally expensive.

### 2.5.5 Advantages vs. disadvantages of high-Level problem-algorithm interaction features

After summarizing the latest trends in high-level problem-algorithm interaction features, Table 2.3 presents their advantages (pros) and disadvantages (cons).

Table 2.3: Advantages vs. disadvantages of high-level problem-algorithm interaction features.

| **High-Level p-a features** | **Advantages** | **Disadvantages** |
|---|---|---|
| Learned by CNNs | - More applicable for 2D and 3D problems<br>- Does not require any feature engineering | - Black-box features<br>- Loss of high-dimensional information by representing the data as a 2D fitness map<br>- Depends on the algorithm portfolio used for learning |
| TransOptAs | - Does not require any feature engineering | - Black-box features<br>- Depends on the algorithm portfolio used for learning<br>- Requires algorithm execution for obtaining algorithm performance labels used for supervised training |
| Based on performance | - Facilitates algorithm comparison through performance vectors | - Biased to the selected portfolio of benchmark problems |
| Based on Shapley values | - Encodes interactions between problem features and algorithm performance<br>- Used to find similar algorithm behaviors with the assumption that the predictive models behave similarly | - Depends on the selected problem features portfolio<br>- Depends on the selected benchmark problem instances<br>- The dependencies on the surrogate model |
| via Knowledge Graph | - Encodes interactions between problem features and algorithm performance by also involving the graph neighborhood | - Depends on the data stored in the graph and the graph embedding method used for learning |
| via graph neural networks | - Encodes interactions between problem features, algorithm configuration, and algorithm performance by involving the graph neighborhood | - Depends on the data stored in the graph and the GNN method used for learning |
| Based on fANOVA | - For each problem, they encode the contributions of modules and their interactions to the final performance of an algorithm. | - Depends on the available data from the pool of different configurations. |
| Based on SHAP | - For each problem, they encode the contributions of each separate module to the final performance of an algorithm. | - Depends on the available data from the pool of different configurations. |

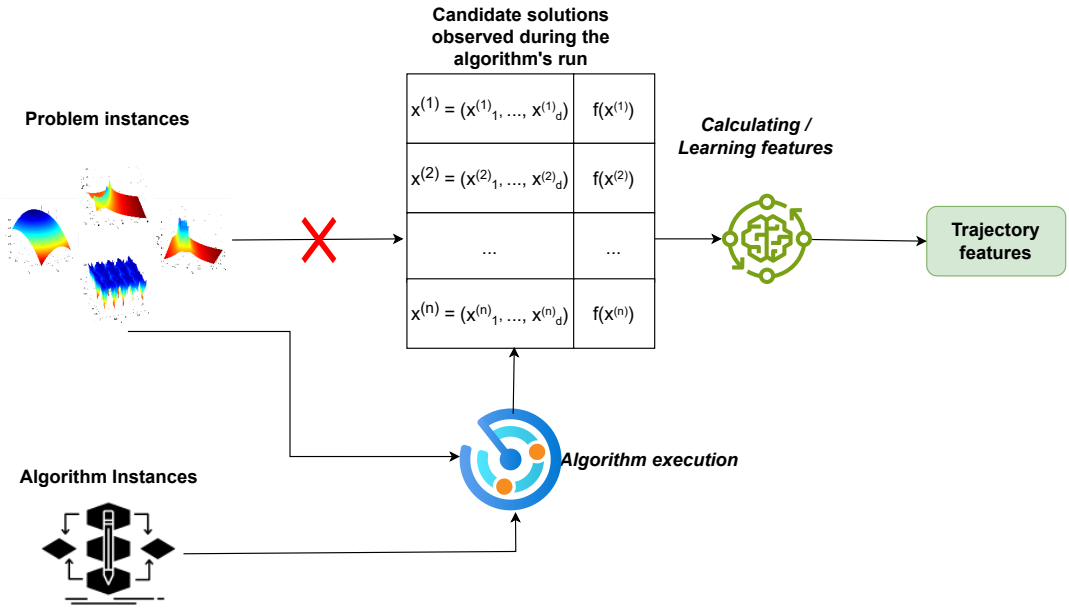



Figure 2.4: The general pipeline of calculating problem-algorithm trajectory features.

## 2.6 Trajectory-based Features that Capture Problem-Algorithm Interaction

The previously mentioned approaches for problem landscape features compute low-level features by generating an artificial set of candidate solutions using a sampling strategy that spread well across the whole decision space. They are hence not connected to the actual set of candidate solutions explored by an optimization algorithm instance *during* its optimization process. To address this limitation, several studies have undertaken automated AS on a per-instance and per-run basis by calculating *trajectory-based features* that represent the algorithm behavior during its run. Below, we are going to provide more details about different trajectory-based approaches for learning features that capture problem-algorithm interaction. Figure 2.4 presents the general pipeline of calculating problem-algorithm trajectory features.

### 2.6.1 Trajectory-based features based on internal algorithm parameters

Some optimization algorithms contain internal parameters that are adapted throughout the optimization process, such as the step size, the evolution path, the conjugate evolution path, and the square root of the diagonal of the diagonal matrix holding the eigenvalues of the covariance matrix. Building on features suggested in [103], the work [104] studied the use of time-series features (using the *tsfresh* [105] library) on top of these algorithm-internal parameters. These time series features were used to identify the configuration of modular CMA-ES variants.

### 2.6.2 Trajectory-based ELA features

Trajectory-based ELA features are computed from a set of candidate solutions that are derived from the entire algorithm trajectory, rather than from a standard sampling technique

(such as Latin Hypercube Sampling, random sampling, etc.) that is applied to the objective function. This means that the samples are extracted from the populations (i.e., sets of candidate solutions and their corresponding function values) that are generated/observed by the optimization algorithm during its execution. The trajectory-based ELA features have been first proposed in [25], under the name of *Adaptive Landscape Analysis*. In that study, the ELA features are computed using samples from the distribution that the algorithm CMA-ES uses to sample its solution candidates. The approach has been evaluated for fixed-budget performance prediction [106] of the CMA-ES [35], as well as *per-run* AS with warm-starting [107], where the features extracted from the trajectory of an initial optimization algorithm instance are used to determine whether to switch to a different algorithm instance. We need to highlight here that the trajectory-based ELA features use a subset of all candidate solutions explored by the algorithm instance during its run, without considering the iteration in which these solutions were generated. These features do therefore not capture the longitudinality of the solutions that are observed within the iterations of the algorithm execution. Another study that uses the trajectory-based ELA has been conducted to analyze the benefits of predicting the switching between different algorithm instances during the optimization trajectory run [108]. It has been also shown that they can be used for dynamic selection of the acquisition function of Bayesian Optimization algorithms, improving over default static choices [109].

### 2.6.3 DynamoRep features

DynamoRep features [57] are capturing the problem-algorithm interaction using simple descriptive statistics extracted from the populations explored by the algorithm in each iteration of its execution on a problem instance. These features are presented in more detail in Chapter 5.

### 2.6.4 Opt2Vec features

The trajectory-based ELA and DynamoRep features are designed to analyze and represent the entire trajectory of an optimization run. However, an important aspect is the information contained within a small, specific segment of the optimization trajectory, such as a particular timestamp, iteration, or population. This is especially crucial when it comes to optimizing dynamic algorithms efficiently. For this purpose, Opt2Vec [58] features propose the usage of autoencoders to encode the information found in the interaction between an optimization algorithm and optimization problem into an embedded subspace. These representations take the populations considered by an optimization algorithm instance in each iteration of its execution, scale the candidate solutions and objective function values, and identify a subset of informative populations. The subset of informative populations is obtained by calculating the Forbenius norm on the differences of the matrices representing consecutive populations of the algorithm's execution, and discarding one of the populations if the Forbenius norm is smaller than a predefined threshold. The proposed approach is designed to capture features for parts of the optimization problem's search space that the algorithm explores at a specific timestamp or iteration during the optimization process. The Opt2Vec features are evaluated for the task of classifying problems from the CEC benchmark suite. In particular, 30 CEC problems in three different dimensions (10, 30, 50) are used. Different dimensionalities of the same problem are treated as different problems, resulting in 90 problem classes used in the classification task. The Opt2Vec features achieve classification accuracies over 80% for the recognition of the problem being solved and its dimension.

### 2.6.5 Iterative-based ELA features

An indirect outcome of the Opt2Vec study are the iterative-based ELA features, which involve calculating the ELA features on each population separately, and concatenating the population representations to generate a representation for the entire trajectory. In this way, the representations would capture the dynamics and behavior of the algorithm on the problem landscape, however, the dimension of the generated representation (i.e., number of features) grows linearly with the number of iterations for which the algorithm is run.

### 2.6.6 Local optima networks

Local Optima Networks (LONs) [110] serve as a simplified model for representing discrete (combinatorial) fitness landscapes, where local optima are nodes and search transitions are edges defined by an exploration search operator. They represent the number of local optima in the landscape, as well as their distribution and connectivity patterns. Monotonic LONs are a variant of LONs where transitions between local optima are considered only if fitness is non-deteriorating. To account for neutrality in the level of local optima transitions, i.e., connected components in the monotonic LONs that have the same fitness value, compressed monotonic LONs have been proposed [111]. A compressed local optimum can be described as a grouping of interconnected nodes within the monotonic LONs, all sharing the same fitness value. While originally proposed for combinatorial problems, LONs have been adapted for the continuous domain in [111]. In this case, a Basin-Hoping algorithm [112] is used to identify the local optima, and the monotonic LONs and compressed monotonic LONs are adapted for the continuous problems.

Compressed monotonic LONs have been utilized to visualize the 24 BBOB problem classes across different problem dimensions [113]. Network metrics are also calculated for each compressed monotonic LON, and dimensionality reduction techniques are employed to classify and compare these problems. The findings reveal that compressed monotonic LONs exhibit varied representations that are linked to the inherent properties of the problem instances and their dimensionality. Network metrics prove particularly crucial for multimodal problems in higher dimensions, where compressed monotonic LONs become too intricate for meaningful visual interpretation.

Similar to LONs are the Search Trajectory Networks (STNs) [114], which are novel tools to study and visualize how population-based algorithm instances behave in continuous spaces. Inspired by LONs, which map the nodes to local optima, in STNs the nodes focus on different states from the optimization trajectory not limited to local optima. The edges signify the progression between these states. This expansion enhances network-based models' utility for understanding heuristic search algorithms. However, they have not yet been evaluated as possible sources for learning trajectory-based features.

### 2.6.7 Probing trajectories

In a recent study, researchers introduced an algorithm selector method that utilizes short probing trajectories generated by running an initial algorithm on a problem instance for a brief period [115]. These trajectories are then used to determine either the current fitness or the best-so-far fitness, which are tracked across predefined sequential iterations of the initial algorithm. Following this, time-series features are extracted using the *tsfresh* library. These time-series features are then used as input for a Random Forest classifier to train the algorithm selector. Additionally, the raw probing trajectories are fed into a Rotation Forest classifier for the same purpose. This approach shows promising results comparable to those achieved using trajectory-based ELA features extracted from observed candidate solutions and a Random Forest model.

### 2.6.8 ClustOpt

A representation learning and visualization methodology is proposed [116], in which solution candidates explored by the algorithm are clustered, and the evolution of cluster memberships across iterations is tracked. In this way, a dynamic and interpretable view of the search process is provided. Additionally, two metrics—algorithm stability and algorithm similarity—are introduced to quantify the consistency of search trajectories across runs of an individual algorithm and the similarity between different algorithms, respectively. This methodology is applied to a set of ten numerical metaheuristic algorithms, through which insights into their stability and comparative behaviors are revealed, thereby enabling a deeper understanding of their search dynamics.

### 2.6.9 Advantages vs. disadvantages of trajectory features.

After summarizing the latest trends in problem-algorithm trajectory features, Table 2.4 presents their advantages and disadvantages.

Table 2.4: Advantages vs. disadvantages of trajectory features.

| **Trajectory features** | **Advantages** | **Disadvantages** |
|---|---|---|
| Based on internal parameters | - Capture the behavior of the algorithm | - Lack of comprehensive comparison of different time series features<br>- Do not enable a comparison of different algorithms (with different configurations), limited to comparing configurations of the same algorithm |
| Trajectory-based ELA | - Info about the interaction across problem and algorithms (personalization) | - Does not capture the longitudinal aspect of solutions within algorithm iterations |
| Iterative-based ELA | - Info about a single timestamp of the optimization process, can easily be combined with ML models that will capture the longitudinality of the search process | - ELA features are sensitive on small sample sizes |
| DynamoRep | - cheap to compute<br>- Despite lower computational cost, DynamoRep features yield similar performance as ELA features that are calculated at each iteration of the algorithm's execution | - Limited expressiveness<br>- Representation size grows with number of iterations and problem dimension, may require dimensionality reduction as preprocessing step |
| Opt2vec | - Capture features specific to parts of the search space explored at a particular iteration<br>- Takes into consideration the optimization problem dimension since one model is used for all problem dimensions. | - Depends on the data used to train the autoencoder<br>- Black-box features |
| LON and variants | - Useful for visualization purposes | - Very costly to compute |
| Probing trajectories | - Potential to be utilized for per-run AS | - Requires suitably chosen probing algorithms<br>- Difficult to transfer results from one probing algorithm to another |
| ClustOpt | - Useful for algorithm behavioral analysis | - Depends on the clustering algorithm and its parameters |

## 2.7 Machine Learning Studies that Utilized Optimization Features

Table 2.5 provides an overview of the research focused on problem, algorithm, and trajectory-based features and their applications in problem classification, AS, performance prediction, and complementarity of benchmark suites. The table is structured into three horizontal parts, one for each of these categories. Each row represents a specific class of features. The columns are divided into three primary aspects. The first aspect identifies the learning tasks in which the features are evaluated, including problem classification, AS, performance prediction, and visualization/complementarity analysis. The second and third aspects concern the collection of problem instances used in the research, which may originate from established benchmark suites such as BBOB, CEC, or Nevergard, or be generated by problem instance generators.

We omit a detailed description of each study presented in the table, as most have been previously discussed, and instead provide a single example to illustrate how the table can be interpreted. For instance, the study [107] explores per-run AS using trajectory-ELA features and algorithm features derived from internal algorithm parameters. Accordingly, in the first part of the table, this study is listed in the cells (Trajectory-ELA, AS) and (Internal Algorithm Parameters, AS). For additional context, the same reference [107] appears in the second part of the table, corresponding to the BBOB and Nevergrad benchmark suites. These are the benchmark suites where the learning tasks from the first part of the table are tested.

The table results indicate that the majority of studies have utilized ELA problem landscape features with BBOB benchmark problem instances. This prevalence is expected, as developing an ML model for problem classification, algorithm performance prediction, or AS benefits from having multiple problem instances per problem class that vary by shifting or scaling. This variation allows for training the ML model on one set of problem instances and testing it on another set that differs only in shift or scale. However, using other benchmark suites, such as the CEC benchmark suites, remains challenging. These suites require the development of zero-shot ML models [117] because each problem class is represented by only a single instance.

Recent trends indicate the emergence of novel problem landscape features, such as TLA, DoE2Vec, and TransOpt, which have demonstrated predictive accuracy comparable to ELA features in problem classification tasks. Additionally, there is a growing interest in trajectory-based features, which capture information about the interaction between an algorithm instance and a problem instance.

A common limitation across all studies is the restricted generalization of the developed ML models to other benchmark suites such as CEC and Nevergrad [63], or to instances generated by problem generators. This is evident from the predominant use of the BBOB benchmark suite for learning and evaluation. To address this limitation in the future, it is essential to incorporate problem instances from diverse benchmark suites and various problem generators (which remain underexplored) into a representative learning set to enhance generalization.

Table 2.5: Summary of works using problem, algorithm, or trajectory-based features and their applications in the domain of continuous SOO.

| | **Learning tasks** | | | | **Benchmark suites** | | | **Problem generators** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Features | Problem Classification | AS | Performance Prediction | Visualization / Complementarity | BBOB | CEC | Nevergrad | ISA | GP | TR | Affine | GKLS |
| **Problem landscape features** | | | | | | | | | | | | |
| ELA | [14], [71], [75], [80], [118] | [1], [8], [9], [25], [56], [64], [119], [120] | [10], [19], [25], [33], [48]–[51], [64], [102], [121]–[123] | [6], [22], [65], [66], [94], [118] [7], [30], [33], [60], [81], [122] | [1], [6]–[8], [10], [22], [25], [30], [33], [48]–[51], [56], [60], [64]–[66], [71], [75], [80], [81], [94], [102], [118]–[120], [122], [123] | [6], [7], [10], [19], [66], [81], [121], [123] | | [30] | [65] | [8], [60], [94], [120] | [9], [22], [33], [120] | [66] |
| TLA | [13], [14] | [13] | | | [14] | | | | | | | |
| Fitness Map + CNNs | [90] | [89] | | | [89], [90] | | | | | | | |
| Point Cloud Transformer | [90] | | | | [90] | | | | | | | |
| DoE2Vec | [11] | | | | [11] | | | | | [11] | | |
| TransOpt | [95] | [120], [124] | | | [95], [120] | | | | | [120], [124] | [120] | |
| DeepELA | [12] | [12] | | | [12] | | | | | | | |
| Random Filter Mappings | [99] | [99] | | | [99] | | | | | | | |
| **Algorithm features** | | | | | | | | | | | | |
| Source Code | | [42] | | | | | | | | | | |
| **High-level problem-algorithm interaction features** | | | | | | | | | | | | |
| Performance | [43] | | | [43] | [43] | | | | | | | |
| Explainable Prediction Models | | | [10], [51] | | [10], [51] | | | | | | | |
| Internal Algorithm Parameters | [104] | [107] | [104] | | [104] | | | | | | | |
| Graph embeddings | | | [44] | | [44] | | | | | | | |
| fANOVA | | | | [46] | [46] | | | | | | | |
| Shapley values | | | [102] | | [102] | | | | | | | |
| **Trajectory-based features** | | | | | | | | | | | | |
| Trajectory-ELA | [25] | [106]–[108], [125] | | | [25], [106]–[108], [125] | | [107] | | | | | |
| DynamoRep | [126] | | | | [126] | | | | | | | |
| Opt2Vec | [58] | | | | | [58] | | | | | | |
| Iterative-ELA | [58] | | | | | | | | | | | |
| LON | | | | [111], [113] | [111], [113] | | | | | | | |
| Probing trajectories | | [115] | | | [115] | | | | | | | |
| ClustOpt | | | | [116] | [116] | | | | | | | |

# Chapter 3

# Problem Landscape Features Based On Transformer Models

Transformer-based architectures have achieved remarkable success across a wide range of domains, including natural language processing [16], [127], [128], computer vision [17], and time-series modeling [129]. Their widespread adoption can be attributed to several key properties, such as the ability to model long-range dependencies through self-attention [15], process entire input sequences simultaneously [15], and their suitability for efficient parallel training on large-scale datasets [130]. These advantages have established transformers as a dominant paradigm for representation learning in modern machine learning research.

Motivated by these successes, in this chapter, we investigate the potential of developing transformer-based problem landscape features using models trained solely on raw samples of the objective function. This approach enables models to autonomously learn rich and task-relevant representations of the problem landscape without any intermediary feature calculation step. This investigation is guided by hypothesis H1, which posits that landscape features learned directly through transformer models can accurately predict algorithm performance when operating solely on raw samples from the objective function.

We demonstrate the supervised training of a transformer architecture for continuous SOO problem instances on two different tasks:

*i)* problem classification - In Section 3.1, the transformer model is trained on the task of classifying BBOB problem instances into one of the 24 problem classes. In further chapters, we refer to this model as the TransOpt model [95].

*ii)* algorithm performance prediction - In Section 3.2, the transformer model is trained to predict algorithm performance. We refer to this model as the TransOptAS model [124].

## 3.1 TransOpt

In this section, we present the TransOpt model [95], where the transformer is trained for the BBOB problem classification task. BBOB problem classes provide a natural source of supervision for learning meaningful embeddings of optimization problems. Each class corresponds to a family of functions with distinct landscape characteristics (e.g., separability, conditioning, global structure), which are directly relevant to algorithm behavior. By training a transformer to predict BBOB classes from raw samples, the model is encouraged to encode these underlying structural properties in its latent space. The resulting embeddings capture problem-relevant information that can be transferred to downstream tasks such as AS or performance prediction, without relying on hand-crafted features.

### 3.1.1 Methodology

In this subsection, we describe the methodology for training the TransOpt model, starting with the data and the sampling process. We then describe the model architecture and the training process, and finally, we explain how the comparison to the ELA features is performed.

#### 3.1.1.1 Data

The BBOB [18], [131] suite contains 24 continuous SOO problem classes. Each problem class can further have multiple problem instances, which represent a transformation (for instance, scaling or shifting) of the original problem class. We use the first 999 instances of each problem class to train the transformer model to predict one of the 24 problem classes. We use problem classes of dimensions 3, 5, 10, and 20. The data is available at `https://doi.org/10.5281/zenodo.7930604`.

#### 3.1.1.2 Sampling

The first step of the proposed methodology involves the generation of samples from which the problem representations are calculated. The samples are obtained using Latin Hypercube Sampling [24], with a sample size of $50d$ and $100d$, where $d$ is the problem dimensionality. The objective values of the samples (y-values) are scaled to be in the range [0,1], while the candidate solutions (x-values) are kept in their original range of [-5,5], since this range is fixed in the BBOB.

#### 3.1.1.3 Model architecture

The heart of the original transformer architecture is its encoder-decoder structure, where both the encoder and the decoder are composed of multiple identical blocks. In our case, we only use the encoder part of the transformer architecture, which consists of a multi-head self-attention module and a position-wise feedforward network.

Transformer models can contain several layers, i.e. several stacked submodules in the encoder or decoder. Each layer consists of a multi-head attention component, a feed-forward component, and some normalization and residual connections. The multi-head attention component can contain several parallel attention subcomponents i.e. attention heads, with each head computing a different attention score and output for the same input. The outputs of all heads are then concatenated and projected to form the final output of the multi-head attention layer. The number of heads controls how many different aspects of the input can be attended to simultaneously.

Since transformer models have become widely used across different domains and our implementation of the encoder is essentially identical to the original, we omit an exhaustive background description of the transformer architecture and refer readers to the original work [15]. One thing to note is that the original transformer architecture is invariant to the ordering of the input data. In other ML fields such as natural language processing or time series analysis, where the ordering of the input data (i.e. the order of words in a sentence or the time component of a time series) is important, a positional encoding is added to the input embeddings to model the sequence information. In our case, since the samples of an optimization problem do not have an inherent ordering, the architecture should be kept order-invariant, so we do not add a positional encoding.

Figure 3.1 features the full model architecture and an example of a single training sample. The input to the model are the samples from the optimization problems. A single training instance of the ML model is a matrix of shape $sx(d+1)$, where $s$ is the number

of problem instance samples, and $d$ is the dimensionality of the problem. In this case, the second dimension of the input matrix is $d + 1$, since the value of the objective function is also included.

The encoder produces an embedding of size $e$, which is a specified model parameter, for each of the given samples from the optimization problem, i.e. it outputs a matrix of the shape $[s, e]$. In order to obtain a single, flat representation of the problem, we calculate several descriptive statistics on the matrix obtained from the encoder. In particular, we calculate the minimum, maximum, mean, and standard deviation of the representations of the samples produced by the encoder. Concatenating the vectors obtained using each descriptive statistic, we obtain the representation of the problem, which is then fed into a classification head, which produces the class of the problem. The classification head contains a linear layer with a Rectified Linear Unit activation, a dropout layer, and a linear layer which performs the classification into 24 problem classes.

#### 3.1.1.4 Model training

The model is trained with a cross-entropy loss, the Adam optimizer [132], and a maximal learning rate of 0.001. The training is executed for a maximum of 200 epochs, with an early stopping mechanism that prevents overfitting by terminating the training process if the validation loss does not observe a decrease of at least 0.001 for five epochs. We use stratified 10-fold cross-validation to evaluate the model, where in each fold, 81% of all instances from each class are used for training, 10% are used for testing, 9% are used for validation.

#### 3.1.1.5 Comparison to ELA features

The proposed representation learning method is compared to the performance of the most commonly used ELA features for the problem classification task. We adopt the conventional methodology of using ELA features and a Random Forest (RF) model. Please note that we have also performed the comparison with a neural network similar to the classification head of the transformer model. However, we present the results of the RF model since it outperformed the neural network model. This means that we are not comparing the feature representations with the same model, but looking at the ELA and transformer models as completely different approaches.

We compute ELA features from the following categories using the *flacco* R library [69]: *basic*, *disp*, *ela_distr*, *ela_level*, *ela_meta*, *ic*, *nbc*, and *pca*. In this way, we calculate a total of 93 features, on which the model is trained to predict the problem class. We also perform experiments where the model is trained on each group of the previously mentioned features. In case of missing values in the calculated ELA features, we discard features that have more than 10% missing values in the training set, while the remaining missing values are imputed with the mean value of the feature in the training set. The model is evaluated using stratified 10-fold cross-validation, where the splitting of problem instances into folds is consistent with the splitting used for the transformer model.

### 3.1.2 Results

In this subsection, we present the results of the comparison of the performance of the TransOpt model to the performance of the ELA features. We then analyze the errors produced by the TransOpt model.

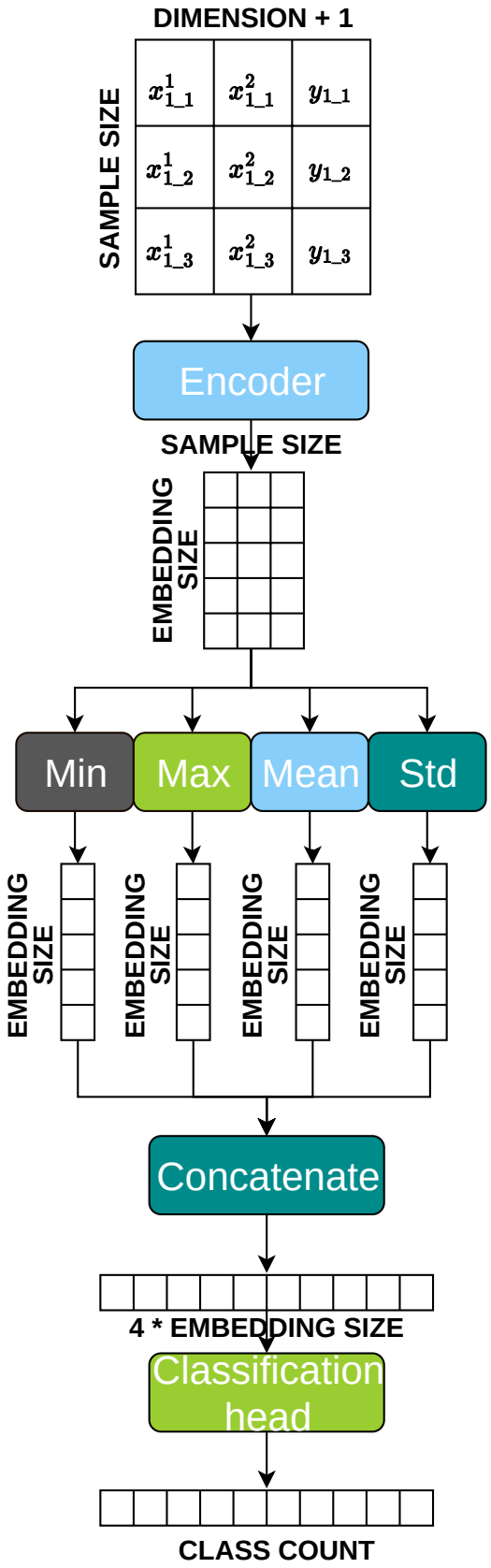



Figure 3.1: Model architecture with example inputs being samples of two-dimensional optimization problems.

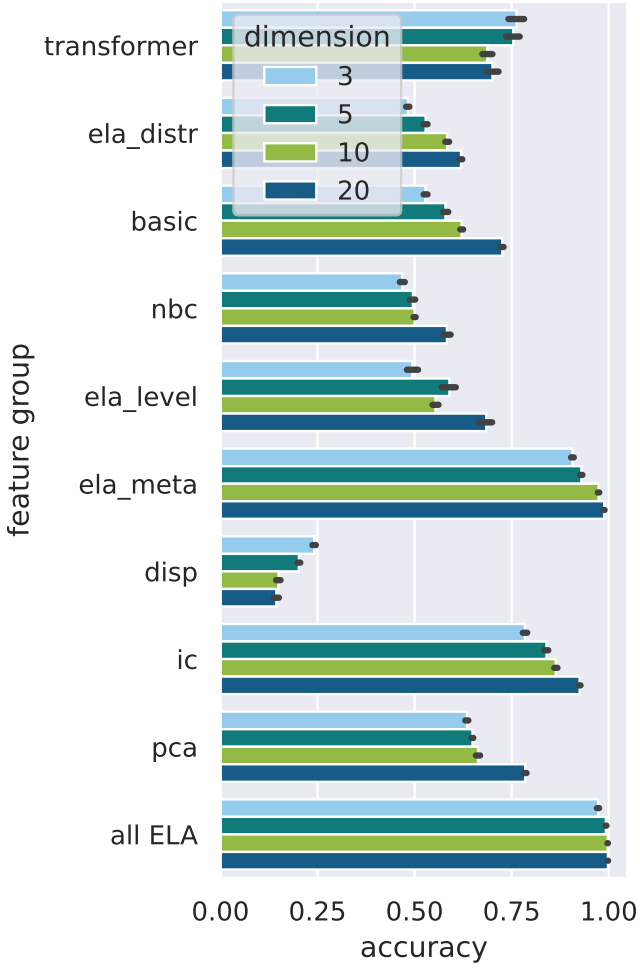



Figure 3.2: Comparison of the accuracy obtained by the transformer model to the ELA features for the classification of problem instances into the 24 classes of problems in the BBOB benchmark

#### 3.1.2.1 Comparison to ELA features

We compare our results to the performance of the ELA features. Figure 3.2 shows a comparison of the problem classification accuracies obtained by the ELA features to the accuracies obtained by the transformer model. In this case, the accuracies obtained by the transformer model using all aggregation functions are labeled as *transformer*, the accuracies of the model trained using all ELA feature groups are labeled as *all ELA*, while the remaining labels refer to individual ELA feature groups. The different colors refer to different problem dimensions.

As can be seen, the best results are obtained using all of the ELA features. As far as individual feature groups are concerned, the *transformer* features are ranked third for problems of dimension 3, 5, and 10, and fourth for problems of dimension 20. The *ela_meta* feature group is ranked first for all problem dimensions. Since transformers are invariant to rotation and other transformations, it is somewhat expected that they are outperformed by ELA on some instances, since some ELA feature groups contain features that are specifically designed to detect such properties.

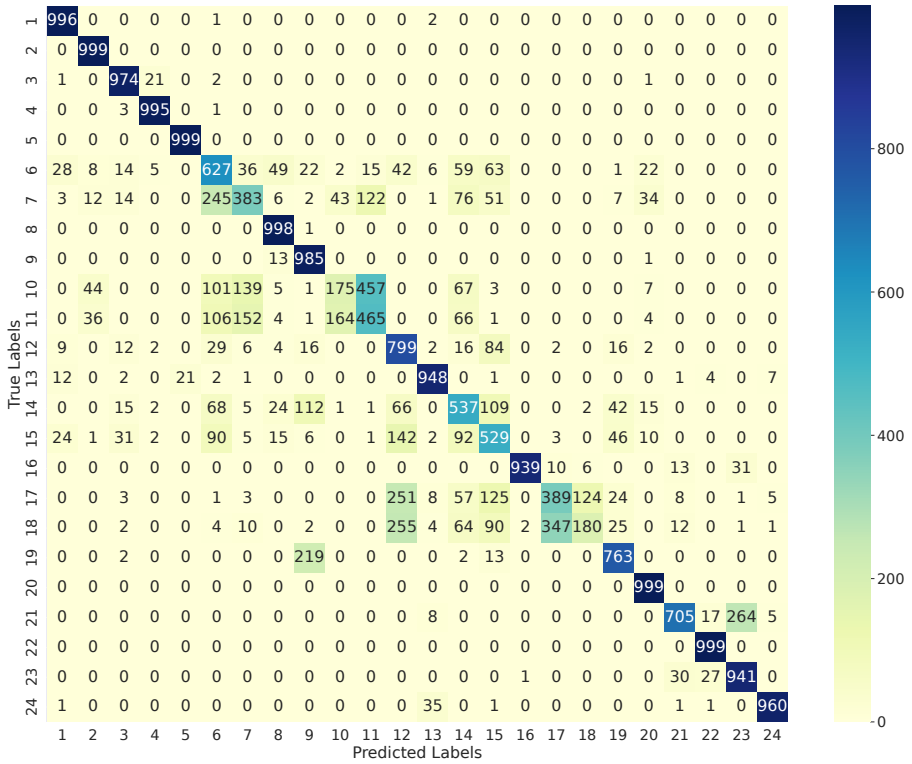



Figure 3.3: Confusion matrix for the classification of problem instances of dimension 3 into the 24 classes of problems in the BBOB benchmark.

#### 3.1.2.2 Error analysis

Figures 3.3 and 3.4 show the confusion matrix for the classification of problem instances of dimensions 3 and 20, respectively. We can see that some of the same problem classes are being confused for both dimensions. Problem classes 6 and 7 (Attractive Sector Function and Step Ellipsoidal Function), 12 and 17 (Bent Cigar Function and Schaffers E7 Function), 12 and 18 (Bent Cigar Function and Schaffers E7 Function, moderately ill-conditioned), 17 and 18 (Schaffers E7 Function, and Schaffers E7 Function, moderately ill-conditioned), and 9 and 19 (Rosenbrock Function, rotated and Composite Griewank-Rosenbrock Function F8F2) are commonly mistaken in both dimensions. On the other hand, problem classes 1 and 13 (Sphere Function and Sharp Ridge Function), 16 and 23 (Weierstrass Function and Katsuura Function), 21 and 22 (Gallagher's Gaussian 101-me Peaks Function and Gallagher's Gaussian 21-hi Peaks Function) are only mistaken for one another in dimension 20, and problem classes 21 and 23 (Gallagher's Gaussian 101-me Peaks Function and Katsuura Function) are only mistaken in dimension 3.

## 3.2 TransOptAS

In this section, we introduce the TransOptAS model [124], which is trained to predict algorithm performance from the samples of an optimization problem. To the best of our knowledge, this is the first attempt at training a transformer architecture for the AS task directly. We demonstrate the training of a transformer architecture for AS on a set of continuous SOO problem instances which are generated using a random function generator [21]. We perform experiments using two sets of algorithm portfolios, one containing

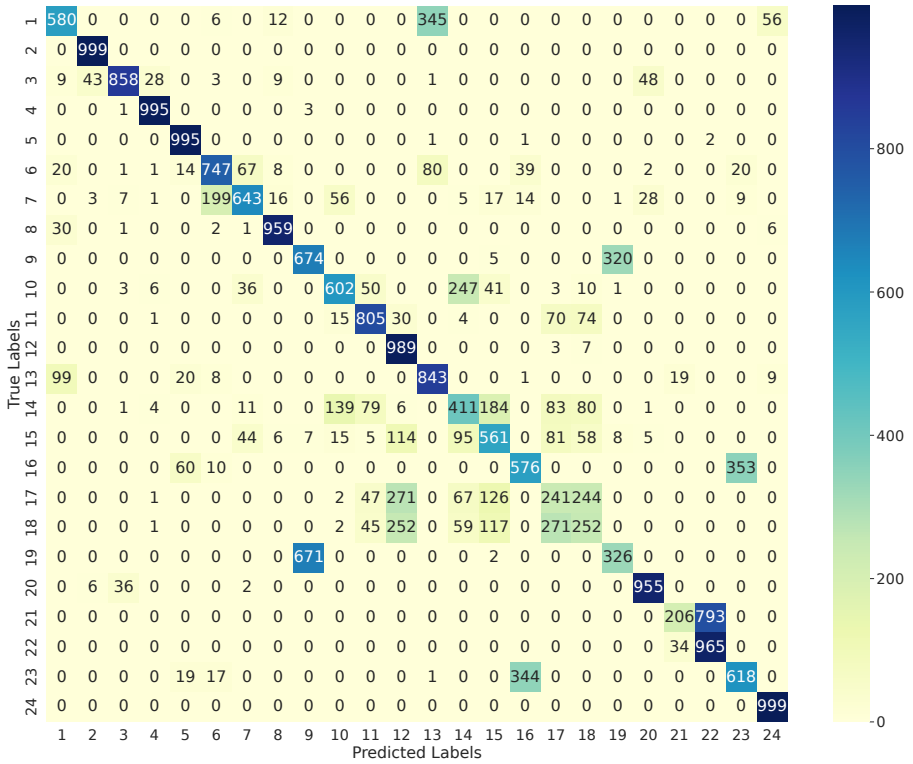



Figure 3.4: Confusion matrix for the classification of problem instances of dimension 20 into the 24 classes of problems in the BBOB benchmark.

four different algorithms, and another one containing twelve different configurations of the Particle Swarm Optimization (PSO) [36] algorithm. We compare the results to a conventional model using the ELA features. We show that a transformer model can successfully be trained for AS using only raw samples of the objective functions, provided there is complementarity in the algorithm portfolio. The proposed approach provides comparable results to the common ELA approach, with the ELA model outperforming the transformer model with differences in the range 0.02-0.05.

**Reproducibility:** The code and data used in the experiments are available at `https://github.com/gjorgjinac/TransOptAS`.

### 3.2.1 Methodology

We first present the problem portfolio, i.e., the set of problem instances used for training the AS model and the sampling process. We then present the algorithm portfolios and the algorithm performance metric. We explain the model architecture, the AS loss and the model training. Finally, we outline the comparative analysis between the proposed methodology and the conventional ELA-based AS approach.

#### 3.2.1.1 Problem portfolio

We generate 5,000 random problem instances of dimension 3 and 10, using the random function generator proposed in [21]. The generator constructs a tree representation by randomly combining mathematical operands and operators from a predefined pool, where each operand and operator have a specific probability of being selected. We use the Python

implementation of the random function generator, which was originally implemented in Matlab and was re-implemented in Python in [133].

One limitation of the generator is that it can produce functions that return constant values for all inputs. Since it does not make sense to optimize such functions, we remove them from the initial set of created functions. The generator can also produce duplicate functions or functions with substantial similarities. Including the same (or nearly identical) problem instances in both the training set and the test set can lead to an unfair evaluation. For this reason, we use the SELECTOR methodology [61] to remove duplicate problem instances from the data. This is accomplished in the following way. We first generate a fixed sample of candidate solutions (x-values) with a sample size of $50d$ where $d$ is the problem dimensionality. We then evaluate all of the generated functions in this sample to obtain the objective function values (y-values).

The objective function values are scaled in the range [0,1] within the samples of each problem instance individually. Next, we calculate the Euclidean distance between the objective function values of each pair of problem instances, obtained on an identical sample of candidate solutions. The SELECTOR methodology then involves the creation of a graph where nodes represent problem instances, and each pair of nodes is connected with an edge if the Euclidean distance of their objective function values is less than 0.01. The Maximal Independent Set algorithm [134] is then executed on the graph, producing a set of problem instances where no duplicates are present. This procedure resulted in 2,638 $3d$ functions and 2,696 $10d$ functions used for the model training.

Please note that the use of SELECTOR is not a required step in our proposed methodology, but it is merely a way to mitigate the issue of duplicate problem instances present in the data, to allow a fair evaluation and to reduce the inherent bias that originates from the data quality.

Another thing to point out is that we are using a random function generator instead of established benchmarks such as the BBOB, since training a transformer architecture from scratch requires a large number of diverse problems, and 24 problem classes with relatively similar problem instances within each problem class do not meet these requirements.

#### 3.2.1.2 Sampling

The first step of the proposed methodology involves the generation of samples from which the problem representations are calculated. The samples are obtained using Latin Hypercube Sampling [24] with a sample size of $50d$, where $d$ is the problem dimensionality. We choose this sample size since it has been shown to work well both for the ELA features [135], and for training the TransOpt model for problem classification. The objective values of the samples (y-values) are scaled to be in the range [0,1], while the candidate solutions (x-values) are kept in their original range. Scaling the y-values can facilitate the training of the transformer architecture, however, this has also been shown to improve generalization of the model to unseen functions when using ELA features [82]. The data is available at `https://doi.org/10.5281/zenodo.7930604`.

#### 3.2.1.3 Algorithm portfolios

To ensure a robust analysis, we use two different algorithm portfolios, for which we train separate AS models. The first portfolio consists of four distinct algorithms, namely Differential Evolution (DE) [136], Genetic Algorithm (GA) [137], Particle Swarm Optimization (PSO) [36], and Evolutionary Strategy (ES) [138]. These algorithms are executed with their default configurations from the *pymoo* library (version 0.6.0), except for the population size, which is fixed to $10d$ for all algorithms. Conversely, the second portfolio includes

12 distinct configurations of the PSO algorithm, with specific parameter settings detailed in Table 3.1. In this table, the *Algorithm* column contains the name of the algorithm configuration, the $w$ parameter denotes the inertia used in each iteration for the velocity update, while *c1* and *c2* denote the cognitive (personal best) impact and the social (global best) impact during the velocity update, respectively. The *adaptive* parameter indicates whether $w$, *c1*, and *c2* are changed dynamically over time, while the *initial velocity* specifies how the initial velocity of each particle is assigned. The population size is kept the same for all algorithms, using its default value of 25 in the pymoo library.

The execution of these algorithms is facilitated through the *pymoo* [139] Python library on all designated problem instances. The algorithms operate within a fixed-budget scenario, with algorithm performance metrics being recorded at budgets of 10, 30, and 50 iterations. For a $3d$ problem, a 50-iteration budget translates to a total of 1,500 function evaluations (50 iterations with a population size of 30). All algorithms use Latin Hypercube Sampling [24] to construct the initial population. We perform ten executions of each algorithm on all problem instances, and capture the overall algorithm performance on each problem instance as described in the following subsection.

Table 3.1: Parameters of the 12 PSO configurations.

| Algorithm | initial velocity | w | adaptive | c1 | c2 |
|---|---|---|---|---|---|
| PSO1 | zero | 0.9 | True | 2 | 2 |
| PSO2 | zero | 0.9 | False | 2 | 2 |
| PSO3 | random | 0.9 | True | 2 | 2 |
| PSO4 | random | 0.9 | False | 2 | 2 |
| PSO5 | zero | 0.2 | False | 1 | 1 |
| PSO6 | random | 0.2 | False | 1 | 1 |
| PSO7 | zero | 0.7 | False | 1 | 2 |
| PSO8 | zero | 0.2 | False | 1 | 1 |
| PSO9 | random | 0.7 | False | 1 | 2 |
| PSO10 | random | 0.2 | False | 1 | 1 |
| PSO11 | random | 0.5 | False | 0.5 | 0.6 |
| PSO12 | zero | 0.3 | False | 0.7 | 1.5 |

#### 3.2.1.4 Algorithm performance metric

We adopt a fixed-budget scenario for assessing the algorithms, which means that we record the best objective function value obtained by each algorithm after a certain number of iterations. Since the random problem instances do not have a known optimum, we cannot use the conventional approach of estimating the achieved precision (the distance of the best-found solution to the optimum) [48]. Instead, we propose a different score that is defined as follows.

Let $y_{arp}$ be the best solution (the lowest objective function value) found by algorithm $A$ in run $r$ for problem instance $p$. We scale $y_{arp}$ by the range of the best solutions found by the other algorithms for the same problem instance and the same initial population (same run). Let $b_{rp}$ and $w_{rp}$ be the best and worst solution found for problem instance $p$ in run $r$ by any of the algorithms under comparison, respectively. Then, the scaled best objective function value achieved by algorithm $A$ in run $r$ on problem instance $p$ is given by:

Table 3.2: Example of the calculation of the algorithm performance metric for one problem instance.

| run | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| Objective value of best solution found (Per run) | | | | |
| 1 | -15.10 | -14.18 | -14.95 | -15.03 |
| 2 | -15.09 | -14.13 | -14.37 | -14.16 |
| 3 | -15.12 | -13.94 | -14.89 | -15.07 |
| Normalized Algorithm Score (Per run) | | | | |
| 1 | 0.00 | 1.00 | 0.16 | 0.08 |
| 2 | 0.00 | 1.00 | 0.75 | 0.97 |
| 3 | 0.00 | 1.00 | 0.19 | 0.04 |
| Final Algorithm Score (Median across all runs) | | | | |
| | 0.00 | 1.00 | 0.19 | 0.08 |

$$s_{arp} = \frac{y_{arp} - b_{rp}}{w_{rp} - b_{rp}} \tag{3.1}$$

This score captures how much the best solution found by algorithm $A$ in run $r$ was better than the other algorithms executed with the same initial population. We calculate the final algorithm score $s_{ap}$, capturing the performance of algorithm $A$ on problem instance $p$ across all runs of the algorithm, by simply taking the median of all $s_{arp}$.

Table 3.2 demonstrates an example of how the algorithm performance scores are calculated for a single problem instance and a single budget. We should note that within the example, each algorithm is executed three times, so we have three runs, however, in reality, we are executing the algorithms ten times on each problem instance. The table is divided in three parts, each one showing a different step of the process of calculating the algorithm performance metric. The first part of the table shows the objective function values of the best solutions found by each algorithm after a fixed budget of function iterations. In the second step, these values are then normalized within each run, by performing a min-max normalization column wise. This means that for each run, the best algorithm obtains a score of 0, while the worst algorithm obtains a score of 1. In the example, the best algorithm is A1, and the worst is A2. The final algorithm score is calculated by taking the median of the normalized algorithms scores for all of the runs. Please note that in some cases where the algorithm performance is not very stable between different runs, it is possible to get final algorithm scores where none of the algorithms have a score of 0 or 1.

We choose this performance metric for several reasons. Unlike the multi-class classification approach, which predicts only one a single best algorithm, this approach considers the performance of all algorithms, avoiding the problem of penalizing a model for predicting one of two algorithms with similar performance. In contrast to the ranking approach, which assigns an integer score to each algorithm based on its performance, this approach reflects a fine-grained difference in performance, allowing algorithms with similar best-obtained values to have similar scores.

#### 3.2.1.5 Model architecture

Figure 3.5 features the full model architecture and an example of a single training sample. The input to the model are the samples from the optimization problems. A single training

Table 3.3: An example of calculation of the misranking score and the loss for two problem instances.

| instance | True scores | | | Predicted scores | | | Loss |
|---|---|---|---|---|---|---|---|
| | A1 | A2 | A3 | A1 | A2 | A3 | |
| 0 | 0.00 | 1.0 | 0.3 | 0.00 | 0.2 | 0.3 | 1.00 |
| 1 | 0.05 | 0.2 | 1.0 | 0.40 | 0.0 | 0.7 | 0.85 |

instance of the ML model is a matrix of shape $sx(d+1)$, where $s$ is the number of samples from each problem instance, and $d$ is the dimensionality of the problem. In this case, the second dimension of the input matrix is $d+1$, since the value of the objective function is also included.

The encoder produces an embedding of size $e$, which is a specified model parameter, for each of the given samples from the optimization problem, i.e. it outputs a matrix of the shape $[s, e]$. In order to obtain a single, flat representation of the problem, we take the mean of the obtained matrix. The produced problem representation is then fed into a regression head which predicts the score obtained by each of the algorithms. The regression head contains a linear layer with a rectified linear unit activation, a dropout layer, and a final linear layer with a sigmoid activation, producing a score in the range [0,1] for each algorithm in the algorithm portfolio.

#### 3.2.1.6 Algorithm selection loss

To capture the true performance of the algorithm predicted to be the best by the algorithm selector, we calculate the loss of the AS:

$$loss = \frac{1}{|\mathcal{P}|} \sum_{p \in P} 1 - (s_{sp} - s_{bp}) \tag{3.2}$$

where $s_{sp}$ is the true score of the selected algorithm which was predicted to have the best performance for problem instance $p$, while $s_{bp}$ is the score of the best-performing algorithm on problem instance $p$ (i.e., the virtual best solver) from the raw performance data. This score would get a value of zero if the worst algorithm is predicted to be the best for all problem instances, and a value of one if the best algorithm is predicted to be the best for all problem instances. Please note that the metric is supposed to be maximized, so higher values are better.

Table 4.1 demonstrates how the loss is calculated for two problem instances and three algorithms. For the first instance, the true best algorithm is A1 with a score of 0, and the model predicts it to be the best. The loss score is 1, because the best algorithm is correctly identified. For the second instance, the A2 algorithm is predicted to be the best, however, the true score of the A2 algorithm is 0.2, and the true best-performing algorithm is A1 with a score of 0.05. If one were to use the A2 algorithm instead of the A1 algorithm in practice, the choice of A2 over A1, they would get a result that is 0.15 (0.2 - 0.05) worse than if they were to use the true best algorithm. The loss score in this case is 0.85 (1-0.15).

#### 3.2.1.7 Model training

The model is trained to minimize the sum of the squared errors between the predicted and true scores of the algorithms. The Adam optimizer [132] is used with a maximal learning rate of 0.001. The training is executed for a maximum of 200 epochs, with an

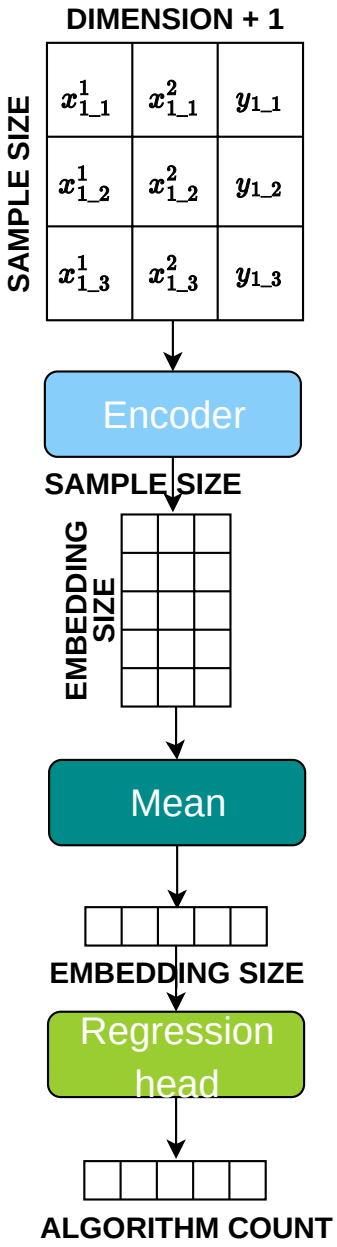



Figure 3.5: Model architecture with example inputs being samples of two-dimensional optimization problems.

early stopping mechanism that prevents overfitting by terminating the training process if the validation loss does not observe a decrease of at least 0.001 for five epochs. The model training is repeated 10 times with different train-validation-test splits. Random splitting is used to split the instances into training, validation, and test data with a ratio 80-10-10. The training data is used to train the model, the validation data is used to determine when to perform the early stopping, while the test data is used to evaluate the model.

The transformer model has several key parameters that can contribute to its performance. The number of layers in the encoder is set to 1, the number of attention heads within each layer is set to 2, the dimension of the produced embeddings is set to 100, the dimension of the keys and values within the attention heads is set to 30. We set these values based on previous experience and experimentation. Parameter tuning will be explored as part of future work.

We train a separate transformer model for each problem dimension, each algorithm portfolio, and each algorithm execution budget.

#### 3.2.1.8 Comparison to the ELA approach

The proposed representation learning method is compared to the performance of the most commonly used ELA features. For this purpose, we adopt the conventional methodology of using ELA features and a RF model trained for AS. We use the RF model since it is the most common choice in the optimization domain [48], [56], [109], and it has good performance on tabular data [140]. To ensure a fair comparison, ELA features are computed on the same set of $50d$ samples that are used to train the transformer model. This means that we are not comparing the feature representations with the same model, but looking at the ELA and transformer models as completely different AS approaches.

We compute ELA features from the following categories using the *flacco* R library [69]: *basic*, *disp*, *ela_distr*, *ela_level*, *ela_meta*, *ic*, *nbc*, and *pca*. In this way, we calculate a total of 93 features, on which the model is trained to predict the problem class. Some of these features contain missing values, which we remove before training the AS model. The model training is repeated 10 times in order to ensure the robustness of the results, where the splitting of problem instances into folds is consistent with the splitting used for the transformer model.

### 3.2.2 Results

In this section, we first give insights into the performance of the algorithm portfolios on our problem portfolio. Subsequently, we show the results of the model training and the comparison to the ELA approach.

#### 3.2.2.1 Algorithm performance

Figures 3.6 and 3.7 show the raw performance scores of the two algorithm portfolios for budgets of 10, 30, and 50 iterations. Each subplot refers to a different problem dimension, the x-axis refers to the budget, while the y-axis shows the name of the algorithm. Figure 3.6 shows the performance of the DE, GA, ES, and PSO algorithms, while Figure 3.7 shows the performance of the different PSO configurations. We can see that for $3d$ problems the algorithms from the first algorithm portfolio provide comparable results, while the GA algorithm provides the best results across all budgets for $10d$ problems. This indicates that in the DE-GA-ES-PSO portfolio, the GA algorithm is a strong single-best-solver for $10d$ problems, making the AS task more challenging. On the other hand, the PSO configurations have similar mean performance scores, with the configurations PSO8, PSO10 and

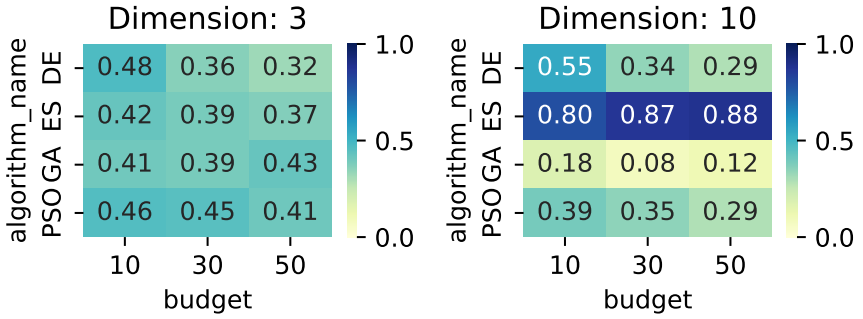



Figure 3.6: Mean algorithm scores for the DE, GA, ES and PSO algorithm portfolio on the 5,000 randomly generated problems in dimension 3 and 10, expressed in terms of the algorithm performance metric defined in Subsection 3.2.1.4.

Table 3.4: Mean and median difference between the losses achieved by the ELA model and the transformer model, aggregated across all budgets.

| Dimension | Algorithm Portfolio | Median | Mean |
|---|---|---|---|
| 3 | DE-GA-ES-PSO | 0.049 | 0.053 |
| | PSO configurations | 0.027 | 0.025 |
| 10 | DE-GA-ES-PSO | 0.023 | 0.029 |
| | PSO configurations | 0.027 | 0.027 |

PSO11 showing somewhat inferior performance for lower budgets, and the configurations PSO2, PSO4, PSO8, PSO10 and PSO11 being inferior on higher budgets.

#### 3.2.2.2 Algorithm selection results

In this subsection, we present the AS results. Figures 3.8 and 3.9 show the loss achieved for the two algorithm portfolios. The loss is depicted on the y axis of each figure, while the x axis represents the algorithm execution budget. The differently colored barplots indicate different models. We compare the transformer model to the RF model using ELA features. We also include a simple dummy baseline, predicting the mean score of each algorithm on the training set.

From Figure 3.8, we can see that the ELA features provide slightly better results than the transformer features for the DE-GA-ES-PSO portfolio. Both the transformer and ELA models outperform the dummy model for $3d$ problems, however, this is not always the case for $10d$ problems. For $10d$ problems, the ELA features outperform the dummy for all budgets, however, for budget 30, the performance is very close to the dummy. The transformer has somewhat worse performance in this case, with the performance being closer to the dummy model than the ELA features. This is due to the fact that for $10d$ problems, the DE-GA-ES-PSO portfolio has a very strong single-best-solver, which makes the dummy model also very strong. On the other hand, for $3d$ problems, there is some diversity in the algorithm portfolio, and this is why the AS works in this case.

Looking at Figure 3.9, depicting the losses achieved for the PSO configuration portfolio, we can see that both the ELA and the transformer features outperform the dummy in both problem dimensions. The ELA features again provide better results than the transformer features, however, the difference between them does not seem to exceed 0.05.

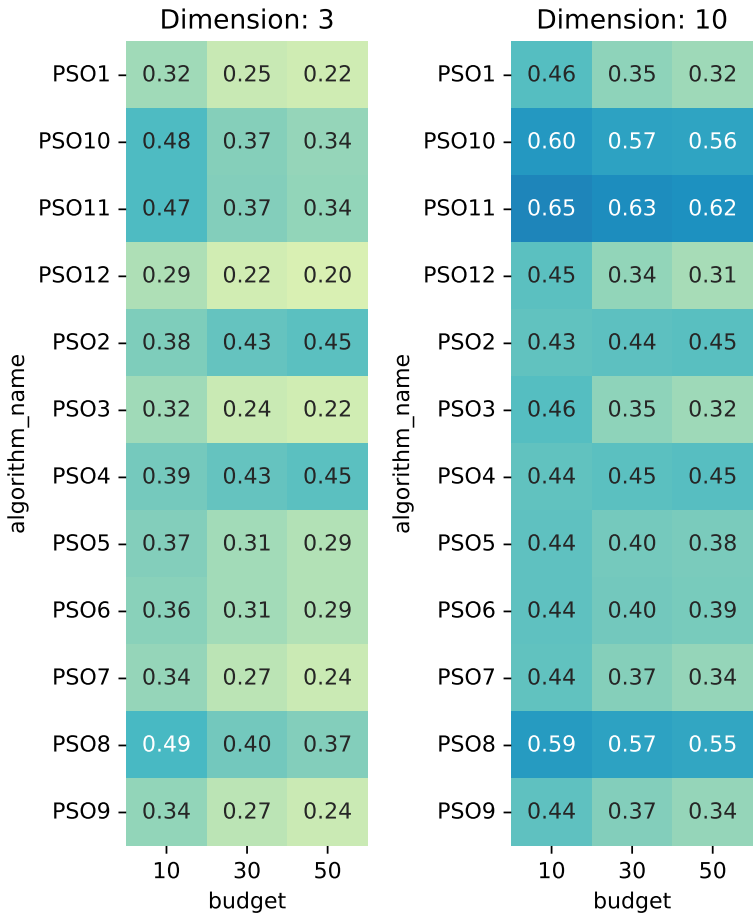
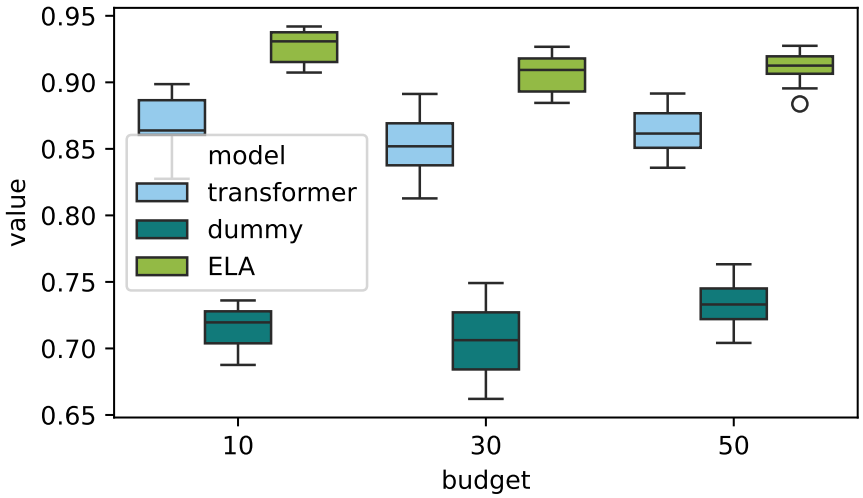
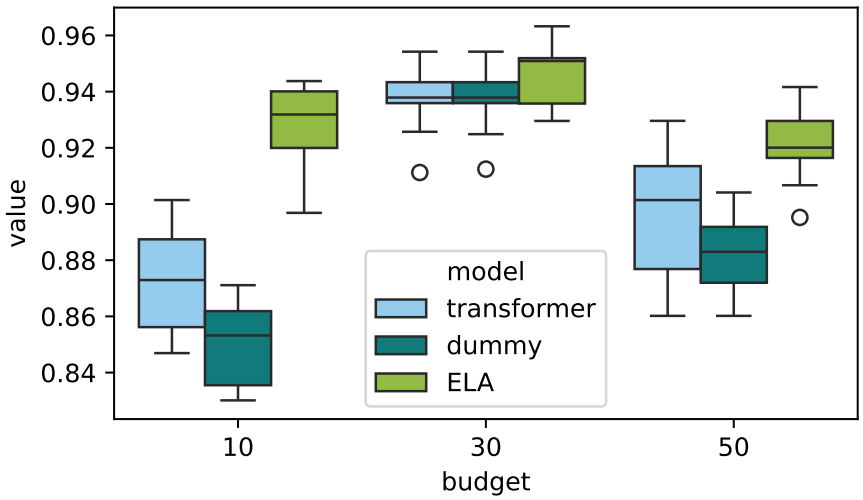



Figure 3.7: Mean algorithm scores for the PSO configuration portfolio on the 5,000 randomly generated problems in dimension 3 and 10, expressed in terms of the algorithm performance metric defined in Subsection 3.2.1.4.

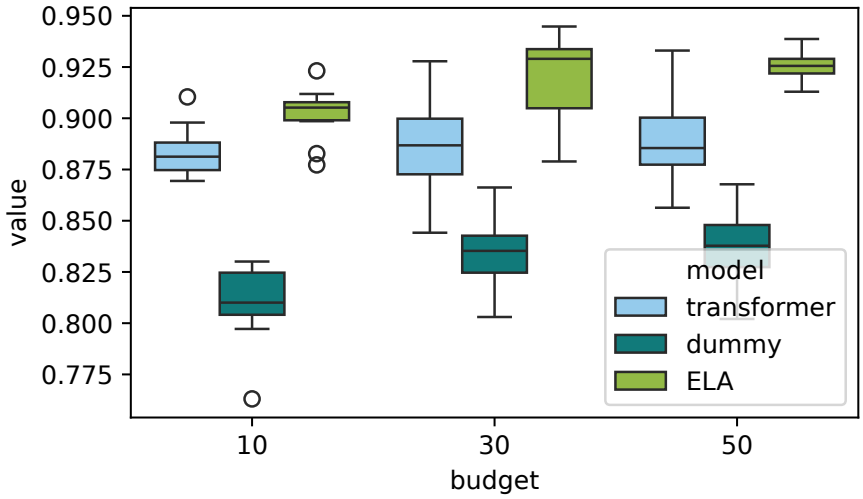



(a) Problem dimension 3

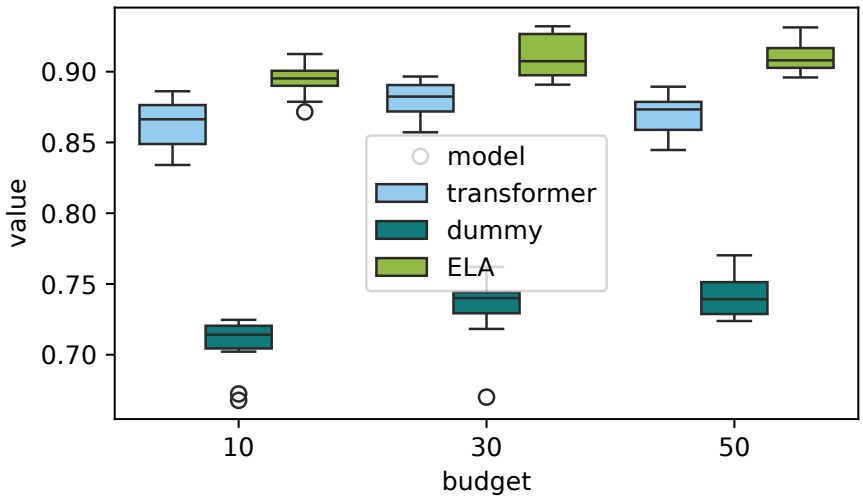



(b) Problem dimension 10

Figure 3.8: Losses achieved by the dummy model, the transformer-based model and the ELA-based model for the DE-GA-ES-PSO algorithm portfolio, expressed in terms of the AS loss metric defined in Subsection 3.2.1.6 (the metric is supposed to be maximized, so higher value is better).




(a) Dimension 3




(b) Dimension 10

Figure 3.9: Losses achieved by the dummy model, the transformer-based model and the ELA-based model for the PSO configuration portfolio, expressed in terms of the AS loss metric defined in Subsection 3.2.1.6 (the metric is supposed to be maximized, so higher value is better).

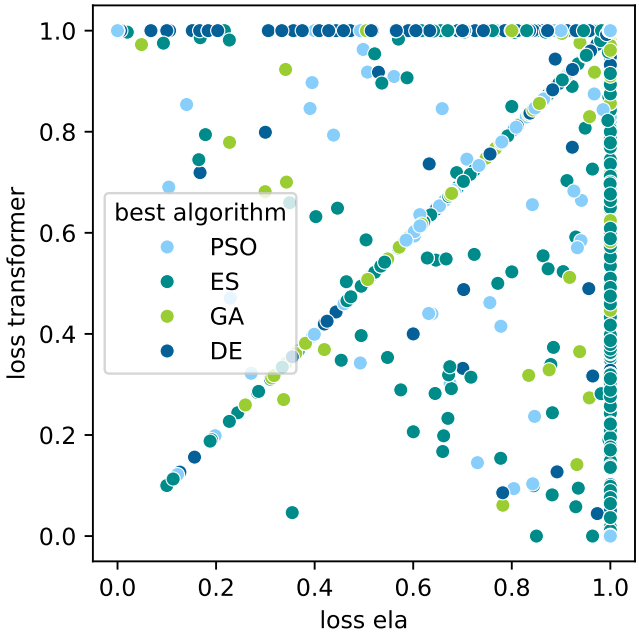



Figure 3.10: Scatterplot depicting the loss (defined in Subsection 3.2.1.6) obtained with the ELA features and the transformer models, for different $3d$ problem instances, for the DE-GA-ES-PSO algorithm portfolio, a budget of 50 iterations.

To summarize the results, Table 3.4 contains the mean and median difference between the losses achieved by the ELA and transformer models. These scores were obtained by calculating the difference between the loss obtained by the ELA model and the loss achieved by the transformer model in each training fold. We then took the mean and median of the obtained differences between the losses for all budgets for a single algorithm portfolio and problem dimension. From the table, we can see that in most cases, the ELA model outperforms the transformer model by approximately 0.02 - 0.05.

In order to analyze where the ELA and transformer models differ in their predictive power, Figure 3.10 depicts the loss obtained with the ELA features and the transformer models, for the AS performed on the DE-GA-ES-PSO on $3d$ problem instances, with a budget of 50 iterations. Each point on the scatterplot represents a single problem instance, while its color reflects the true best-performing algorithm on this problem instance. The horizontal and vertical axes represent the loss obtained by the ELA and transformer models for this problem instance, respectively. From this plot, we can see that there are a lot of problem instances on the diagonal, meaning that the transformer and ELA models produce equivalent loss scores. We can also observe that there is some complementarity between these models, i.e., there are problem instances which are not correctly solved by one model (loss is close to zero), but are correctly solved by the other model (loss is close to one). We should point out that most of the problem instances are concentrated at the point (1,1), meaning that they are correctly solved by both models, though this may not be apparent on the plot, since the points are overlapping. However, we can also observe instances for which both models produce poor predictions, pointing out a need for further improvements.

## 3.3 Conclusion

In this chapter, we demonstrate the supervised training of a transformer architecture for continuous SOO problem instances on two different tasks. First, in Section 3.1, we introduce the TransOpt model, a transformer model trained for the task of problem classification of samples of the problem instances into the 24 BBOB problem classes. Second, in Section 3.2, we introduce the TransOptAS model, trained for the algorithm performance prediction task.

For the problem classification task, the TransOpt model achieves accuracies around 70%-80% for different problem dimensions. Comparing the predictive power of the TransOpt representations to those of individual ELA feature groups, we conclude that the proposed features are among the top three feature groups for different problem dimensions.

For the AS task, our results indicate that the TransOptAS model consistently outperforms a single best solver, provided that there is complementarity in the performance of the algorithms within the algorithm portfolio. These results are in favor of H1, stating that problem landscape features learned directly via transformer models can predict algorithm performance without any intermediary feature calculation step. Although, it is important to note that our comparison with the most commonly used methodology based on ELA features indicates that our proposed transformer approach does not outperform ELA, the differences in predictive power are not substantial, ranging from 0.02-0.05. Interestingly, we observe complementarity in the predictive power of the transformer and ELA models, as there are instances that are correctly solved by one model but not by the other. This introduces an opportunity to also use the TransOpt features to supplement the ELA features, which is explored in further chapters.

# Chapter 4

# Algorithm Selection Generalizability

In most of the available AS studies [8], [10], [19], [64], [107], models are typically trained and tested on the same benchmark suite. Due to the lack of a better resource, the BBOB suite [18] has extensively been used as both a training and testing benchmark, with the generalizability of the performance of the AS rarely being evaluated outside of this suite [8], [10], [19], [64], [107].

Based on the evaluation approaches used in AS studies, the following categorization of evaluation approaches can be made based on how the problem instances are distributed among the training and testing sets:

- **Instance-split evaluation**, which involves training on some instances from all of the problem classes, and testing on other instances from each problem class. This evaluation approach assumes that the test set contains similar instances to the training set since instances of the same problem class are used both for training and testing. This type of evaluation is possible only on the BBOB benchmark suite, where the concepts of a problem instance and problem class are defined. As such, this is the easiest evaluation strategy since the problem instances used in the training and testing phases are relatively similar in terms of features and performance. This evaluation is commonly done in a leave-one-instance-out fashion [56], where a single instance from a problem class is included in the test set, however, multiple instances can also be included. This evaluation type has been criticized as it can produce misleading results for AS, where meta-models achieve high performance due to spurious correlations between features and the target, rather than having genuine predictive capability [141]. This evaluation type has been used in several studies for explainability of AS models [48], [51].

- **Problem-split evaluation**, which involves removing entire problem classes (all of their problem instances) from the training set and using them for testing the ML model. In this case, the ML model may encounter problem instances in the testing set, which are substantially different than the ones used for training, making it a more challenging evaluation scenario. The most common type of the problem-split evaluation in the community is the leave-one-problem-out evaluation, where instances from a single problem are used for testing. Such an evaluation has been explored for performance prediction models on the BBOB and the benchmark from the CEC special session [27] in [19], [121]. In addition, in [19] it was pointed out that when evaluation is done in a problem-split scenario, large errors can be obtained when the AS model is applied to a new problem instance which is not similar to any of the problem instances in the training data. This happens when evaluation is done on the BBOB or CEC benchmarks, since they are limited in the number and diversity

of problem instances, introducing the need of including new benchmark suites in the experimental setup.

- **Evaluating the generalization across benchmarks** - Rather than evaluating the performance within one benchmark, this approach involves the training of an AS method on one benchmark, and evaluating it on a completely different benchmark [8], [10], [64], [142], [143].

Due to the discrepancies in the difficulty of the evaluation in these three contexts, choosing which strategy to use can substantially influence the results [119]. To mitigate this issue and improve the diversity of problems provided in the BBOB benchmark, efforts have been dedicated to generating new problems using problem generators [20], [21], [133] or by combining the original BBOB problems [22], [23].

In this chapter, we investigate the generalizability of AS models using recently proposed problem generators. In particular, Section 4.1 provides a cross-benchmark evaluation of AS models using ELA and TransOpt features. This experimental design supports the evaluation of H2, which posits that the generalizability of an AS model across distinct benchmark suites is influenced by the extent of problem coverage and the underlying performance distribution of the algorithms evaluated on it.

On the other hand, Section 4.2 aims to provide a more comprehensive evaluation of recently proposed features in evaluation settings of increasing difficulty. This study addresses H3, which posits that newly developed landscape features and their combinations outperform the ELA features in the context of AS.

Several works evaluating the generalization of AS models in continuous SOO have already been performed.

In [64], it was observed that an AS trained on the BBOB benchmark has poor generalization to a set of interpolated numerical problems [144]. This observation has been confirmed across multiple studies conducted on different benchmarks.

In [8], all of the previously mentioned types of evaluation have been applied for an AS model which is trained for multi-class classification, where the ELA features are used to represent problem instances. When the model is evaluated using the instance-split evaluation strategy on the BBOB benchmark, a precision of 0.69 and a recall of 0.60 are observed. However, using the problem-split evaluation, the precision score drops to 0.24, while the recall score is 0.32. The authors also investigate the AS generalization across two benchmarks by training the AS on a set of artificial functions and testing on the BBOB benchmark. In this case, a precision of 0.13 and a recall of 0.14 are achieved, indicating no generalization of the model performance across benchmarks. While the ultimate goal of this paper is aligned with ours, the experimental setup is different in that in [8], instead of a multi-class classification model we are training a multi-target regression model to predict a score of algorithm performance that is relative to the best and worst performing algorithms in the algorithm portfolio. The advantage of our approach is that it takes into account the fact that several algorithms may obtain similar results and can quantify the degree to which one algorithm outperforms another. In addition, our study considers a wider set of problem benchmarks involving four benchmark suites (including one that was recently proposed [145]).

The generalization of an AS model using ELA features trained on the original BBOB problem instances and evaluated on their affine combinations has been explored in [9]. The results suggest that while the AS predicts the ranks of algorithms for problem instances similar to the ones observed during training relatively well, its performance substantially declines as the problems instances used for evaluation deviate from the training distribution. In these cases, the model's predictive performance becomes comparable to a baseline model predicting the mean rank of the algorithms in the training set.

In [10], [143], a methodology for assessing how well a predictive model, developed for algorithm performance using one benchmark, can be applied to another. The findings indicate that the patterns of generalizability observed within the problem landscape features are also mirrored in the algorithm performance outcomes.

While most of these studies analyze the generalizability of only the ELA features, our study aims to provide a more comprehensive view of recently proposed features whose generalizability is yet to be investigated.

## 4.1 A Cross-Benchmark Examination of Feature-based Algorithm Selector Generalization in Continuous Single-Objective Optimization

In this section, we first aim to evaluate the generalizability of an AS model across different benchmark suites using the ELA features, as the most commonly used landscape features for continuous SOO problem instances. In this case, the experimental setup involves the use of an entire benchmark is used to train the AS model, and it is evaluated on a completely different benchmark. Secondly, we aim to investigate the degree to which the TransOpt features introduced in Section 3.1 can complement the ELA features in this task. Additionally, we aim to identify the reasons behind the lack of generalizability of AS models when applied to different benchmarks by observing the distribution of feature values across the benchmarks, and the degree to which similarity in problem representations is consistent with similarity in algorithm performance.

For this purpose, we have trained and evaluated three combinations of feature-based AS models that are different in the features used to describe the landscape characteristics of the problem instance (i.e., ELA, TransOpt, ELA+TransOpt). In all cases, a Random Forest (RF) model has been trained in a multi-target regression scenario predicting the performance of four SOO algorithms: DE [136], GA [137], PSO [36], and ES [138]. To test the generalizability of the AS models, four benchmark suites have been utilized: BBOB [18], AFFINE [145], RANDOM [21], and ZIGZAG [20]. Our analysis is focused on the performance of the AS trained on one of these benchmarks, and evaluated on the other ones.

The rest of the section is organized as follows. In Subsection 4.1.1 we present the methodology and experimental setup. Subsection 4.1.2 presents the results of the analysis. Finally, discuss the results in Subsection 4.1.3.

**Reproducibility:** The code and the data for the experiments are available at `https://github.com/gjorgjinac/opt_generalization`.

### 4.1.1 Methodology

In this section, we first introduce the problem portfolios, i.e., the set of benchmarks for training and evaluating the AS. We present the algorithms included in the algorithm portfolio, as well as the performance metric used to compare them. Subsequently, we explain the features used to represent the problem instances. Finally, we present the training of the ML model used for AS, as well as the metrics used to evaluate it.

#### 4.1.1.1 Problem portfolios

Next, the benchmark suites utilized in this study are explained in more detail. These benchmarks were chosen because they had accessible implementations and they allow the

creation of problem instances of different dimensions. We need to note that for all benchmarks, the dimension of the problem has been set at 3 and 10 ($d$ =3 and $d$ =10), where $d$ is the problem dimension.

**BBOB Benchmark** - We use the first 100 problem instances from each of the 24 problem classes from the BBOB benchmark, ending up with 2,400 problem instances.

**Affine Benchmark:** The first five problem instances from the 24 BBOB problem classes have been used to generate affine recombinations as proposed in [145]. The combinations have been done by combining problem instances from different problem classes that have the same instance ID. For example, the first instance of the first problem class is combined with the first instances of the other 23 problem classes. We need to highlight here that instances with different IDs from different problem classes are not combined together. We do this in order to limit the number of generated instances. The recombination is performed with $\alpha$ values of 0.25, 0.50, and 0.75 for all pairs of problem instances. In this way, we obtain 8,280 problem instances. The affine transformations have been generated using the equation suggested in [23], also presented below:

$$\begin{aligned} &F(P_{i,m}, P_{j,n}, \alpha)(x) = \\ &\quad \exp(\alpha\, log(P_{i,m}(x) - P_{i,m}(O_{i,m})) + \\ &(1-\alpha)\, log(P_{j,n}(x - O_{i,m} + O_{j,n}) - P_{j,n}(O_{j,n}))). \end{aligned} \tag{4.1}$$

In this context, $P_{a,b}$ denotes the $b$-th instance of the problem within the $a$-th BBOB problem class. Meanwhile, $O_{a,b}$ signifies the optimum location for the function $P_{a,b}$. The parameter $\alpha$ illustrates the blending extent of the two functions. Every objective function formulated through this equation possesses an optimal solution with an objective value of zero ($P_{a,b}(O_{a,b}) = 0.0$).

**Random Functions:** We generate 10,000 random problem instances using the random function generator proposed in [21] and introduced in Subsection 3.2.1. Please note that there are substantial similarities among some of the objective functions in this dataset. This arises from the way function generation is carried out and is an inherent limitation of the proposed generator.

**ZIGZAG Benchmark:** We generate 5,000 problem instances by randomly initializing the parameters of the four zigzag functions proposed in [20]. The $k$ parameter, controlling the period of the objective function, is randomly sampled as an integer in the range [0,30]. The $m$ parameter, controlling its amplitude, is sampled in the range [0,1]. It has been shown that depending on the $m$, $k$, and $\lambda$ parameters, the generator can create diverse enough objective functions where some algorithms struggle to solve them while others do not.

The rationale for incorporating these benchmark suites is rooted in the diverse needs of the optimization domain, which are designed for different purposes. On the one hand, benchmarks containing carefully designed problems for which we understand their high-level properties and difficulty are better suited for understanding the strengths and weaknesses of different algorithms by observing their behavior on problems with different properties. The BBOB is designed with this goal in mind, while the AFFINE, RANDOM, and ZIGZAG are not. However, the training of an AS model (that involves ML) that will generalize to real-world problems requires a large and diverse set of problems to be used for its training. This introduces a need for a different type of benchmark, which will provide a large coverage of the problem space, with thousands of different problems. Manually designing such a benchmark takes a lot of effort. To the best of our knowledge, there is no benchmark containing real-world problems that are large enough to satisfy the requirements of training an AS model, which is why people are resorting to random problem generators.

#### 4.1.1.2 Algorithm portfolio

The algorithm portfolio for performing AS consists of the DE [136], GA [137], PSO [36], and ES [138] algorithms. The *pymoo* [139] python library is used to run them on the problem instances from all of the benchmarks. The algorithms are executed in a fixed-budget scenario with a population size of $10d$. The performance of the algorithm is recorded at budgets of 10, 30, and 50 iterations. For a $10d$ problem, a budget of 50 iterations is equivalent to a total of 5,000 function evaluations (50 iterations with a population size of 100). All algorithms use Latin Hypercube Sampling [24] to construct the initial population. The rest of the configuration properties of the algorithms are set to the default values provided in the *pymoo* library, version 0.6.0. We perform ten executions of each algorithm on all of the problem instances. The selected algorithm portfolio has been done only for illustration purposes, however, in the future other algorithms can also be involved.

#### 4.1.1.3 Algorithm performance metric

We use the relative algorithm performance metric defined in Subsection 3.2.1.4 to capture algorithm performance. We opt to use such a performance metric for several reasons: *i)* As opposed to using multi-class classification and predicting a single best algorithm, this approach captures the performance of all algorithms, circumventing the issue that a model is penalized for predicting one out of two algorithms with very similar performance. *ii)* In contrast with the ranking approach, where each algorithm is assigned an integer score based on its performance, our approach captures a fine-grained difference in performance, allowing algorithms with similar best-obtained values to have similar scores. *iii)* The reason for using a relative and not an absolute score is that two of the involved benchmark suites (RANDOM and ZIGZAG) do not have a defined optimum for the problem instances.

#### 4.1.1.4 Problem representations

We calculate the following features:

- **ELA Features:** We compute ELA features from the following categories using the *flacco* R library [69]: *basic*, *disp*, *ela_distr*, *ela_level*, *ela_meta*, *ic*, *nbc*, and *pca*. In this way, we calculate a total of 93 features for each problem instance. It is important to note that some of these features contain missing values, which we remove before training the AS model. We calculate the ELA features on a sample of the problem instances obtained with Latin Hypercube Sampling in the range of $[-5,5]^d$, exploring sample sizes of $50d$ and $100d$, where $d$ is the problem dimension.
- **TransOpt Features:** We use the TransOpt architecture introduced in Section 3.1, and train it for the BBOB problem classification task, as defined previously. Once the model is trained, we remove the classification head, freeze the remaining weights, and use the representations generated in the second-to-last layer, before the classification head, to represent the problem instance. These representations are 120-dimensional vectors. We calculate the TransOpt features using the same samples as the ELA features to ensure a fair comparison.
- **Merged:** We also consider the merged representations using both ELA and TransOpt features.

#### 4.1.1.5 Model

We use a Random Forest (RF) model to perform multi-target regression, where the model predicts the performance score of each algorithm. We chose the RF model because it

has good performance on tabular data [140] and it provides feature importance values that help with interpretability. The RF model is executed using the default configuration parameters in the *scikit-learn* [146] Python library version 1.2.2. We do not perform parameter tuning in order to evaluate the impact of the proposed features only, on a fixed model configuration. A separate RF model is trained for each algorithm execution budget, each one predicting the scores obtained by the algorithms after this budget. To ensure the robustness of the results, the training of the RF model is repeated 10 times. In each execution, an entire benchmark is used to train the model, and the evaluation is carried out on the remaining three benchmarks.

#### 4.1.1.6 Algorithm selector evaluation metrics

Next, the evaluation metrics used to validate the AS will be explained in more detail. We have used two metrics: pairwise ranking score and loss.

**Pairwise ranking score** - The pairwise ranking score takes into account on how many problem instances in the test set, the pairs of algorithms are correctly ordered according to their score.

$$PRS = \frac{2}{|\mathcal{A}|\,|\mathcal{A}-1|\,|\mathcal{P}|} \sum_{a_i \in \mathcal{A}} \sum_{a_j \in \mathcal{A}, j>i} \sum_{p \in \mathcal{P}} c(a_j, a_i, p) \tag{4.2}$$

where $P$ is the set of all problems, $A$ is the set of all algorithms, and

$$c(a_j, a_i, p) = \begin{cases} 1 & \text{if } s_p(a_i, a_j, p) = s_g(a_i, a_j, p) \\ 0 & \text{if } s_p(a_i, a_j, p) \neq s_g(a_i, a_j, p) \end{cases} \tag{4.3}$$

where the function $s_p(a_i, a_j, p)$ yields -1, 0, or 1 based on whether the predicted rank of algorithm $a_i$ is lower, equal, or higher than the rank of algorithm $a_j$, respectively, on problem $p$. Similarly, the function $s_g(a_i, a_j, p)$ also provides the values -1, 0, or 1, but in this case, it considers the order of algorithms in relation to the ground truth.

The equation calculates the pairwise ranking score by comparing the predicted rank and the actual rank of every pair of algorithms on every problem. If the predicted and actual rank are the same, it adds one to the score. Then it divides the score by the total number of possible comparisons. This gives us a fraction between 0 and 1 that tells us how often the predicted order of algorithms is correct. A higher score means a better prediction.

We additionally use the Loss metric defined in Subsection 3.2.1.6. Both metrics capture different, complementary aspects of the AS model performance, and we use both in order to have a more extensive evaluation.

Table 4.1 demonstrates how the pairwise ranking score and the loss are calculated for two problem instances and three algorithms. For the first instance, the true best algorithm is DE with a score of 0, and the model predicts it to be the best. The loss score is 1, because the metric is simply "flipped" so we are maximizing both scores. However, the order of the GA and ES algorithms is incorrect for this instance. Since there are in total three unique pairwise combinations of algorithms ((DE,GA), (DE,ES) and (GA,ES)), and two of them are ordered properly, the pairwise ranking score for the first instance is 2/3, i.e. 0.66. For the second instance, the GA algorithm is predicted to be the best, however, the true score of the GA algorithm is 0.2, and the true best-performing algorithm is DE with a score of 0.05. If one were to use the GA algorithm instead of the DE algorithm in practice, the choice of GA over DE, would get a result that is 0.15 (0.2 - 0.05) worse than if they were to use the true best algorithm. The loss score in this case is 0.85 (1-0.15). The pairwise ranking score is 0.33, because only the algorithms GA and ES are correctly ordered.

Table 4.1: An example of calculation of the pairwise ranking score and the loss for two problem instances.

| instance | True scores | | | Predicted scores | | | pairwise ranking score | Loss |
|---|---|---|---|---|---|---|---|---|
| | DE | GA | ES | DE | GA | ES | | |
| 0 | 0.00 | 1.00 | 0.30 | 0.00 | 0.20 | 0.30 | 0.66 | 1.00 |
| 1 | 0.05 | 0.20 | 1.00 | 0.40 | 0.00 | 0.07 | 0.33 | 0.85 |

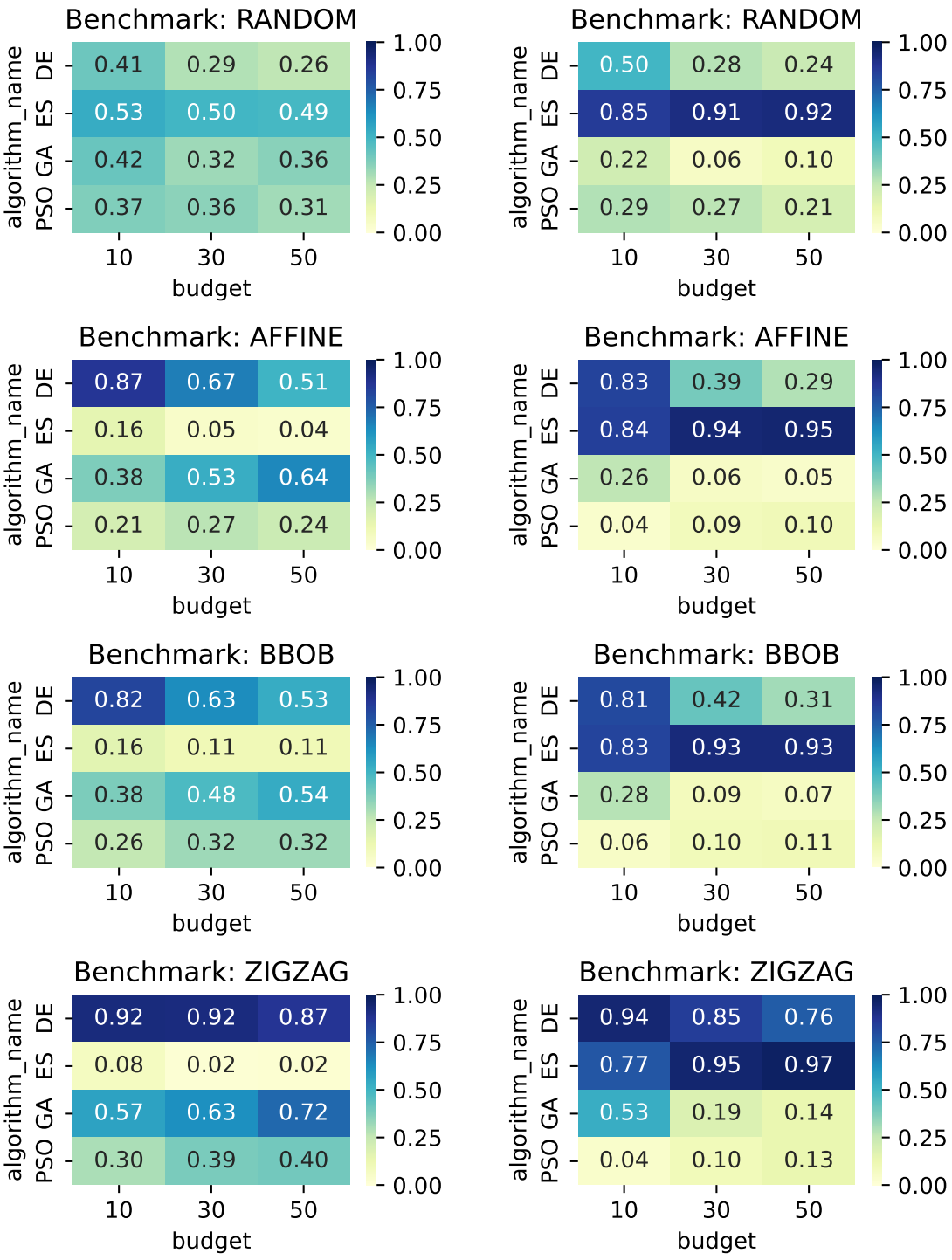
### 4.1.2 Results

First, we analyze the performance of the algorithms across all of the benchmarks. We then evaluate the generalization of the AS models, trained and evaluated on different benchmarks. Subsequently, we aim to identify the reasons behind the lack of generalization between some benchmarks. Specifically, this involves comparing the distributions of the feature values of the benchmarks, analyzing the benchmarks' coverage of the problem landscape, and analyzing the alignment of the problem landscape features and algorithm performance. Our analysis is divided in the following topics:

- Subsection 4.1.2.1 gives a short overview of the performance of the four algorithms on the different benchmarks.
- Subsection 4.1.2.2 presents the results of the AS performed with the different types of features, focusing on identifying whether the feature-based models (ELA and TransOpt) outperform a simple dummy model, and comparing the results obtained with the ELA and TransOpt features.
- Subsection 4.1.2.3 examines the distributions of feature values, to identify if different ranges of feature values are responsible for the lack of generalization which is sometimes observed when training the AS model is done on one benchmark, but evaluated on another one.
- To gain further insights into how similar one benchmark is to another, Subsection 4.1.2.4 analyzes the benchmark's coverage of the problem landscape. More specifically, we perform a clustering of the problem instances from all benchmarks and observe whether problem instances from different benchmarks lie in the same clusters.
- Finally, Subsection 4.1.2.5 investigates the ability of both feature types to capture algorithm performance. In particular, this is done by analyzing whether problem instances with high similarity according to landscape features have similar algorithm performance.

#### 4.1.2.1 Analysis of the raw algorithm performance data

Figure 4.1 depicts the mean scores of the algorithms obtained with a different execution budget. Subfigure 4.1a refers to the $3d$ problem instances, while Subfigure 4.1b refers to the $10d$ problem instances. Each subplot contains the results for a different benchmark, each row refers to a different algorithm, while each column refers to a different budget. From the figures, we can see that the performance of the algorithms on the BBOB and AFFINE benchmarks is very similar. The reason for this outcome is that AFFINE problem instances are a recombination of the BBOB problem instances.




(a) $3d$ benchmarks

(b) $10d$ benchmarks

Figure 4.1: Mean score of each optimization algorithm on $3d$ and $10d$ benchmarks, for budgets of 10, 30 and 50 iterations.

For $3d$ problems in the case of BBOB, AFFINE and ZIGZAG benchmark suites, the ES algorithm provides the best results across all budgets. On the other hand, on the RANDOM benchmark, the ES algorithm provides the worst results, however, a much higher complimentarity in algorithm performance is observed on this benchmark, with algorithms having similar performance scores across all budgets.

For $10d$ problems, the GA and PSO algorithms provide the best results on all benchmarks, while the ES algorithm is the worst-performing.

In summary, we find that the BBOB and AFFINE benchmarks rank algorithms in a similar way for both problem dimensions. These two benchmarks have the same single-best solver as the ZIGZAG benchmark for $3d$ problems. For $3d$ problems, the algorithm performance on the RANDOM benchmark differs from the other benchmarks. This becomes relevant in the next section, as it affects the performance of the baseline model against which the feature-based AS models are compared.

#### 4.1.2.2 Generalization of an algorithm selection model across benchmarks

Figure 4.2 depicts the results of the model generalizability assessment for the $3d$ problems. The figure is separated into six subplots, with rows of subplots referring to a different budget used for algorithm execution, and columns referring to the two evaluation metrics. It is important to note that we aim to maximize both metrics, so a higher score (darker value) indicates better performance. Within each subplot, the row labels indicate the training and testing benchmark, with the first one being the training benchmark. The columns refer to different feature types (ELA, transformer or the merged ELA and transformer features) calculated with different sample sizes ($50d$ or $100d$) used for training the RF model. The first column contains the results of a dummy model, which simply predicts the mean score of each algorithm in the training data and only serves as a baseline against which the other models are compared.

**Comparing AS model performance with a baseline model:**

Next, we focus on comparing the performance of the AS model with a simple baseline. **$3d$ analysis:** Looking at the pairwise ranking score (in the left column of Figure 4.2), we can see that the feature-based models outperform the dummy, except for the cases BBOB-ZIGZAG, AFFINE-ZIGZAG, ZIGZAG-BBOB, and ZIGZAG-AFFINE. The TransOpt features commonly outperform ELA in the cases AFFINE-BBOB, AFFINE-RANDOM, ZIGZAG-BBOB, while the opposite is true for BBOB-RANDOM, RANDOM-AFFINE, RANDOM-BBOB and RANDOM-ZIGZAG. For the remaining cases, both feature types provide similar results.

Looking at the loss metric (in the right column of the figure), there are several cases where the feature-based models achieve very good results (above 0.90), but do not outperform the dummy. This indicates that the models perform well, however, the task itself is not very difficult. The reason for this is that when two benchmarks have the same best-performing algorithm (i.e., single best solver), the dummy model is very strong and it is difficult for any of the feature-based models to achieve a better loss. This is the case for the BBOB, AFFINE, and ZIGZAG benchmarks, on which the ES algorithm largely outperforms the others, resulting in very good performance of the dummy model when the training and testing is executed on a combination of these models. In these cases, none of the models can outperform the dummy in terms of the loss metric, or outperform it by a very small margin.

The RANDOM benchmark has a different distribution of the algorithm performance scores compared to the other benchmarks. Consequently, the loss of the dummy model trained or tested on the RANDOM benchmark is somewhat lower, and the feature-based models manage to obtain a better loss. The TransOpt model obtains slightly better

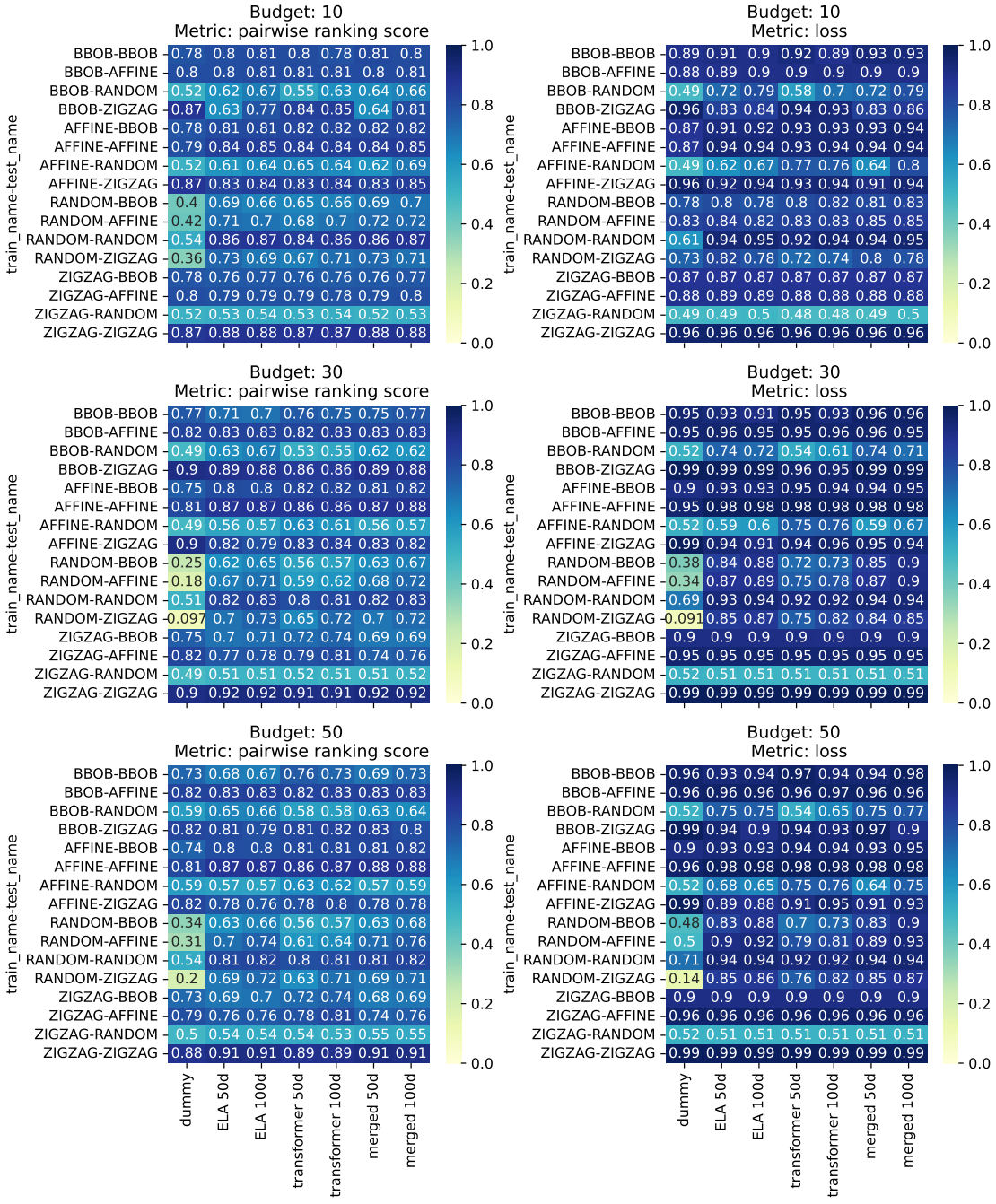



Figure 4.2: Pairwise ranking score and loss achieved by the models for 3$d$ problems.

loss than the ELA features in the AFFINE-BBOB, AFFINE-RANDOM, and AFFINE-ZIGZAG, while the ELA features perform better in the cases RANDOM-ZIGZAG, RANDOM-AFFINE, RANDOM-BBOB, and BBOB-RANDOM.

**10$d$ analysis:** In our experiments for 10$d$ problems, we can again observe the pattern of the dummy outperforming all of the feature-based models for all budgets, which is due to the benchmarks sharing the same single best solver. The TransOpt features outperform ELA in the case AFFINE-RANDOM, while the ELA features outperform the TransOpt features in the cases ZIGZAG-BBOB, ZIGZAG-AFFINE and RANDOM-ZIGZAG. However, in the majority of cases, both feature types provide comparable results.

**Comparison of the feature-based AS models:**

To summarize the obtained results, figures 4.3a and 4.3b show the number of times a specific model outperforms another model type, as well as the mean difference in the loss obtained by the models for 3$d$ and 10$d$ problems, respectively. In this case, we are presenting the generalization results, i.e., when the training and testing benchmark is different, since this is the primary focus of our study. We are only considering the loss metric, since we believe it is more relevant for a practical application of an AS, where we are primarily interested in selecting only the best algorithm, and may not be so interested in the pairwise rankings of the algorithms which is captured by the pairwise ranking score.

The rows in the figures denote which pair of models are compared, while the columns indicate the benchmark that was used for testing the AS. Each cell can have a maximum value of nine, since each model is trained on three different benchmark suites and evaluated on the benchmark suite presented in the column, across three different budgets. The loss results presented are aggregated across different training benchmark suites and different budgets. With this analysis, we aim to analyze which feature types and sample sizes used to calculate the features are better suited for which benchmark.

**3$d$ analysis:** The first row in Figure 4.3a, labeled as "ELA 100d > dummy (count)" indicates how many times (out of 9) the ELA features calculated with a sample size of 100$d$ outperform the dummy model. In contrast, the second row shows the mean difference in the losses obtained with these models.

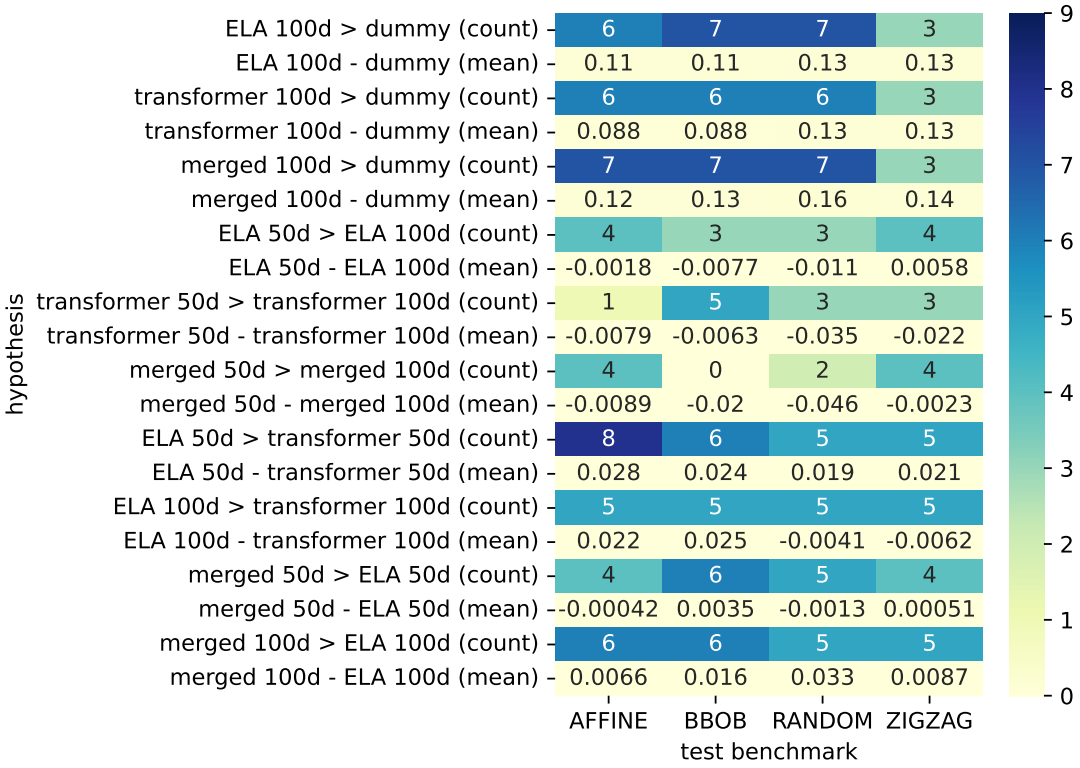
From the first six rows of Figure 4.3a, we can see that the ELA, TransOpt and merged features generally outperform the dummy on the AFFINE, BBOB and RANDOM benchmarks, but not on the ZIGZAG benchmark.

Rows 7-12 indicate that the differences between the losses obtained with a same feature type calculated using different sample sizes are fairly low, meaning that using a larger sample size may not be practically relevant. Evidence of a 100$d$ sample size being preferable over a 50$d$ sample size is only prominent for the TransOpt features applied on the AFFINE benchmark and the merged features applied on the BBOB benchmark, however, the observed mean differences are less than 0.02.

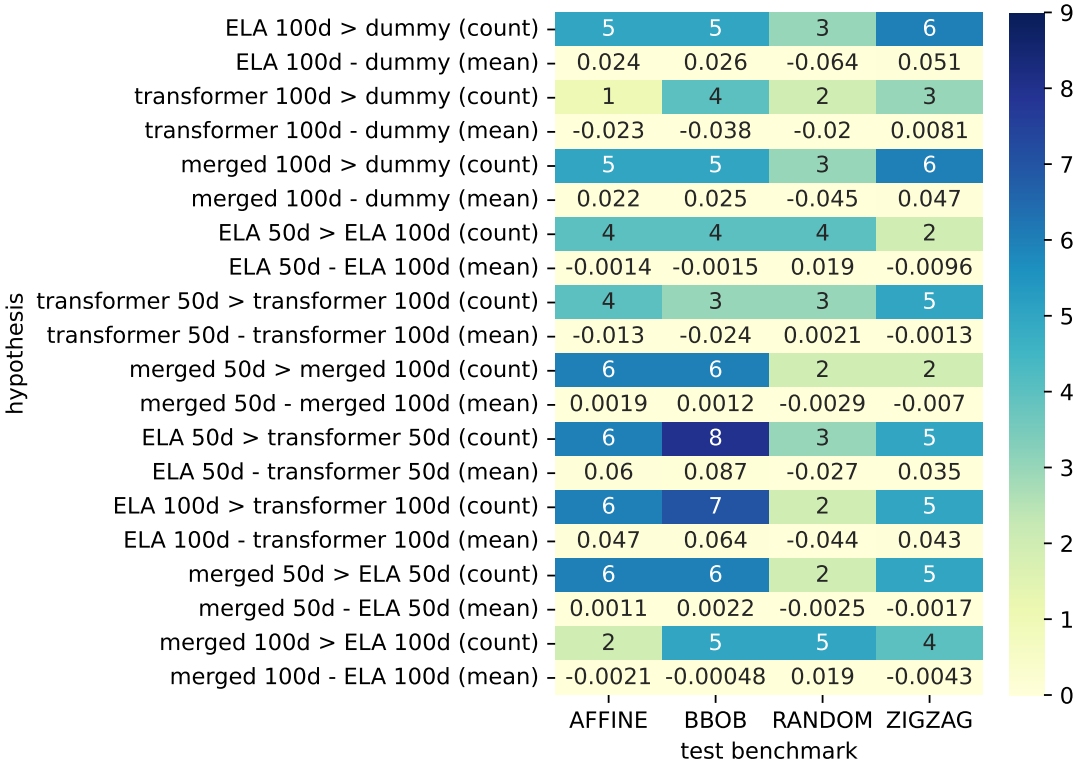
Considering rows 13-16, which analyze whether the TransOpt features obtain better results than the ELA features, we can see that the ELA features consistently outperform the TransOpt features for the AFFINE benchmark with a sample size of 50$d$, while mixed outcomes are obtained for the other benchmarks. The mean difference between the performance obtained with the ELA and transformer features is always less than 0.03.

Finally, the last four rows investigate whether using both the TransOpt features and the ELA features together provides better results than using only the ELA features. In this case, we can again observe mixed outcomes with mean differences in performance being less than 0.03, which does not allow us to make a firm conclusion.

**10$d$ analysis:** Looking at Figure 4.3b, summarizing the results for the 10$d$ problems, we can see that the ELA and TransOpt features do not consistently beat the dummy, with the ELA features beating the dummy more often than the transformer. The performance




(a) Summary results for $3d$ problems.




(b) Summary results for $10d$ problems.

Figure 4.3: Summary results in terms of the loss metric.

obtained with different sample sizes used for feature calculation is consistent with the $3d$ problems, in that the mean differences in performance are quite low, and a strong argument cannot be made for the use of one sample size over another.

The comparison of the ELA and TransOpt features is also consistent with the $3d$ problems, in that ELA features provide better results on the BBOB and AFFINE benchmarks, with mean differences in the range 0.04-0.08. The TransOpt features provide better results on the RANDOM benchmark, while no clear conclusion can be drawn for the ZIGZAG benchmark.

The analysis of whether the merged features outperform using only the ELA features for $10d$ problems shows conflicting outcomes for different benchmarks and sample sizes, with low mean differences, which do not allow for a well-founded conclusion.

To summarize, in this subsection, we show that there are combinations of training and testing benchmarks for which the feature-based models do not outperform the dummy model. The dummy model is particularly strong when the pairs of benchmarks have the same single-best solver. The comparison of the ELA and TransOpt features indicates that the ELA features perform better on the BBOB and AFFINE benchmarks, while the TransOpt features perform better on the RANDOM benchmark. For the ZIGZAG benchmark, conclusive results are not obtained across problem dimensions, regarding the superiority of ELA or TransOpt features.

#### 4.1.2.3 Distributions of feature values across benchmarks

To identify the reasons for the lack of generalization from one benchmark to another (inability of feature-based models to outperform the dummy), we analyze the distributions of the values of the most important features for the models trained on each benchmark. To this end, we first find the top 10 most important features for the AS trained for each budget and each benchmark, by taking the median of the feature importance scores produced by the RF model in each of the 10 training repetitions.

Figures 4.4 and 4.5 show the distribution of feature values (scaled in the range [0,1] for visualization purposes only) for the 10 most important features of the AS trained on each training benchmark for $3d$ problems and a budget of 50 iterations. Figure 4.4 refers to the 10 most important ELA features, while Figure 4.5 refers to the 10 most important TransOpt features. The figures are divided into four subplots, each one containing the top features of the AS trained on a different benchmark. The boxplots in different colors reflect the distribution of feature values across the four benchmarks. In Figure 4.4, we can see that for all benchmark suites some ELA features appear with no variation (e.g., *ela_meta.quad_simple.cond*, *basic.objective_min*). However, we need to clarify, that they do have different values. The impression that they have no variation in the figure is due to the fact that there are a few problems that have extreme values for these features. Including these problems in the figure makes the variation of the feature values of the other problems look negligible. From Figure 4.4, we can see that for some features, the RANDOM benchmark has feature values that are outside of the range of feature values observed in other benchmarks. Such is the case for the features *ic.eps.max*, *ela_meta.lin_simple.cond*, *ela_meta.lin_simple.coef.max*, *ela_meta.quad_simple.cond*, *basic.objective_max*, *basic.objective_min*, *nbc.nn_nb.mean_ratio*, and *ic.m*0. One thing to point out is that some of the features (most apparent for the features *basic.objective_max*, and *basic.objective_min*, measuring the maximum and minimum value of the objective function values in the sample) are very sensitive to the range of the objective function values. A possible solution to ensuring these features have within-distribution values for all benchmarks is to scale the objective function values before computing the ELA features. While scaling the objective value would solve the issue with the features

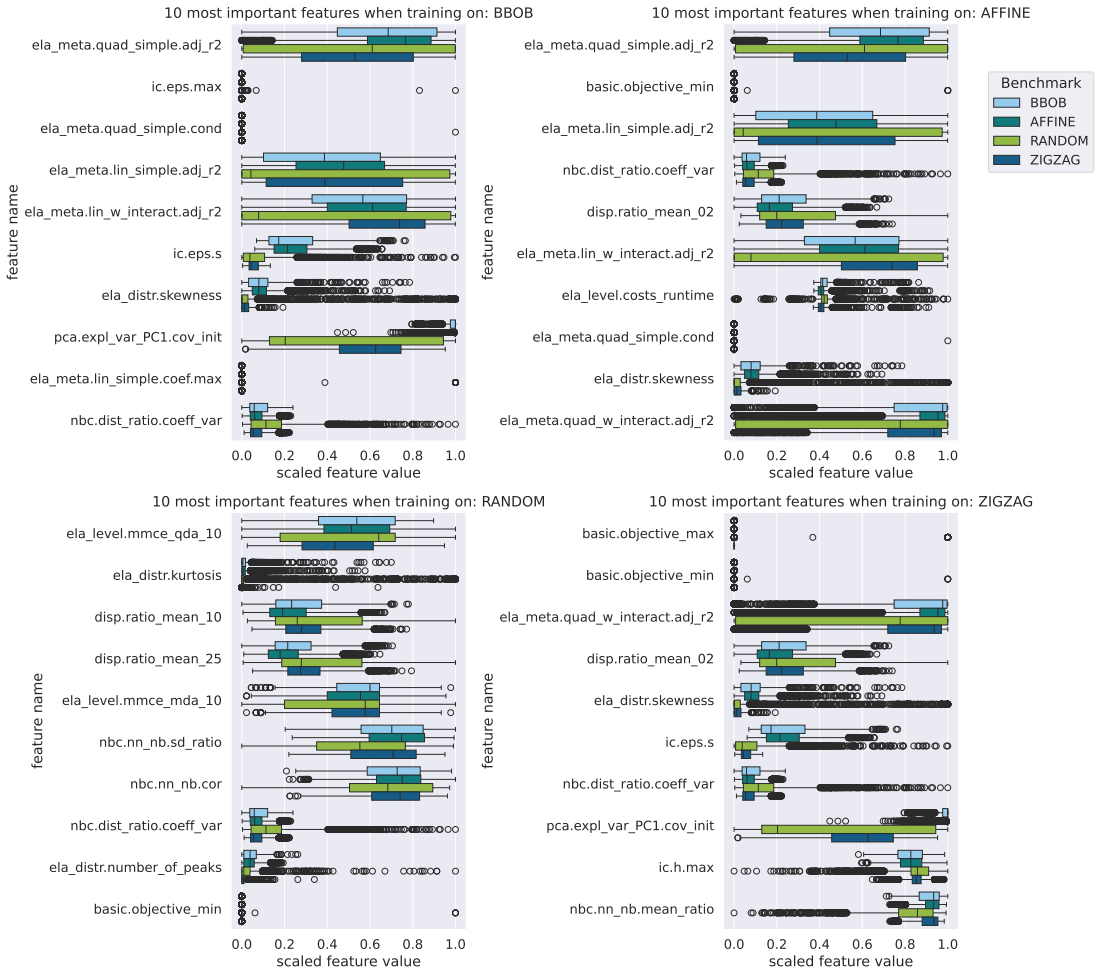



Figure 4.4: Distribution of the values of the top most important ELA features for the AS trained for $3d$ problems and a budget of 50 iterations.

*basic.objective_max*, and *basic.objective_min*, it is not completely straightforward to say whether it would help with the other features. These results are also aligned with results from a recent study [82], where it has been shown that some features are not invariant of shift and scaling if the objective value is not scaled. However, the scaling of the objective function is left for future work as a separate topic and we have performed the experiments with no scaling which is the most common approach presented in the AS studies.

Due to the fact that the TransOpt features are not explicitly defined, but are generated from a neural network architecture, they do not have dedicated names. Because of this, in Figure 4.5, we simply name the features according to their index in the feature vector. In this case, we can also observe that there are several features for which some problem instances from the RANDOM benchmark have values which are out of the distribution of the feature values of other benchmarks. Such is the case, for instance, for features 59, 73, and 63. We should point out here that training a transformer model requires the scaling of the objective function values before computing the features. This means that in this case, the out-of-distribution values obtained for some of the RANDOM problem instances are not simply a result of excessive ranges of objective values, but an indication that there is truly something inherently different about these problems. We can also observe the same for the ZIGZAG benchmark, however, only for feature 49.

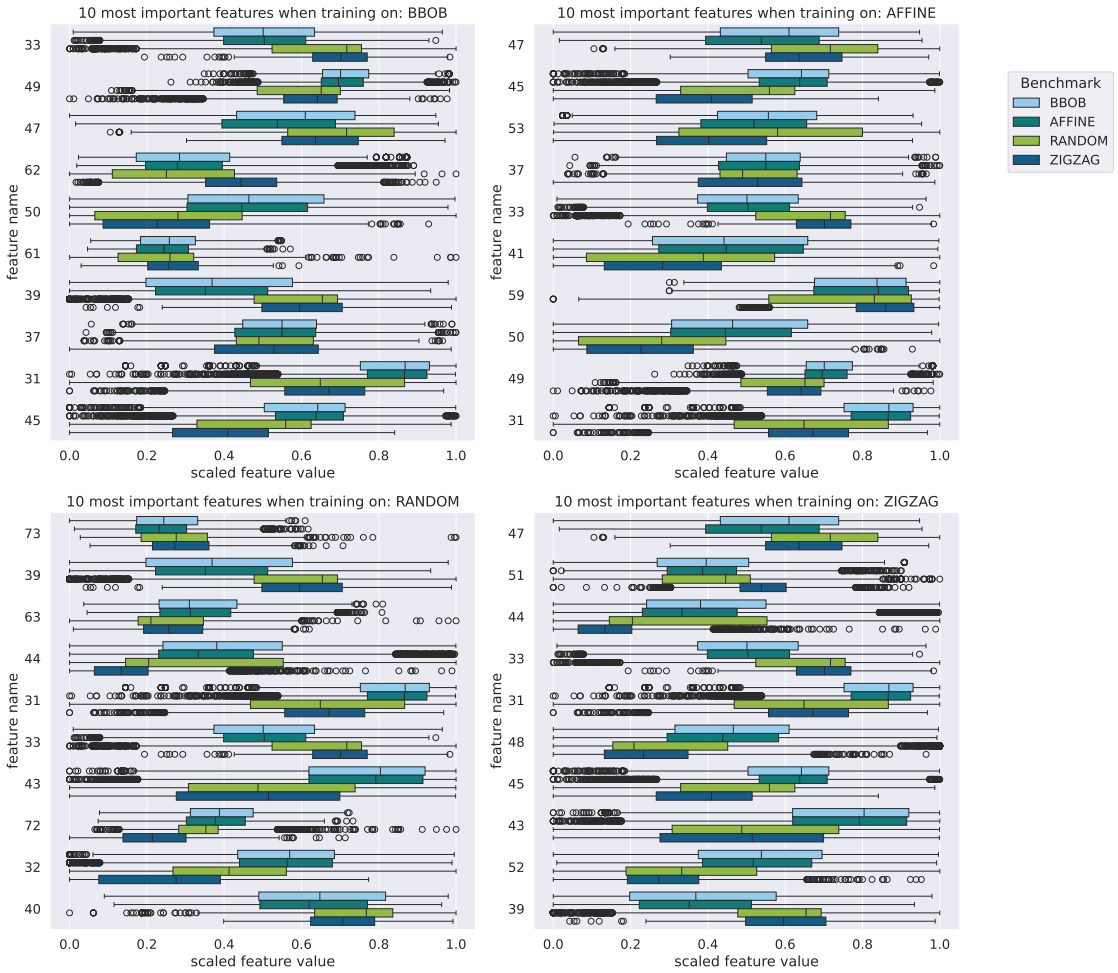



Figure 4.5: Distribution of the values of the top most important TransOpt features for the AS trained for $3d$ problems and a budget of 50 iterations.

In summary, in this subsection, we observe that the RANDOM benchmark has feature values which are outside of the distribution of the other benchmarks, which can explain the poor performance which is sometimes achieved when an AS model is trained on another benchmark, and transferred to the RANDOM benchmark. This behaviour is less prominent in case of the TransOpt features compared to the ELA features, which can also explain why the TransOpt features outperform ELA when the testing is executed on the RANDOM benchmark, as we saw in the previous section.

#### 4.1.2.4 Coverage of the problem landscape

In this subsection, we aim to assess the similarity between problem instances of different benchmarks and analyze the benchmark's coverage of the problem landscape, in order to see if this can explain the generalizability of AS models trained on these benchmarks. To identify the distribution of problem instances from different problem benchmarks, we perform clustering of the problem instances from all benchmarks and analyze the distribution of the benchmarks across clusters. We have performed the clustering twice, using both features (ELA and transformer-based) separately. To this end, we first merge the problem instances from all benchmarks into one set. We then scale the feature representations of

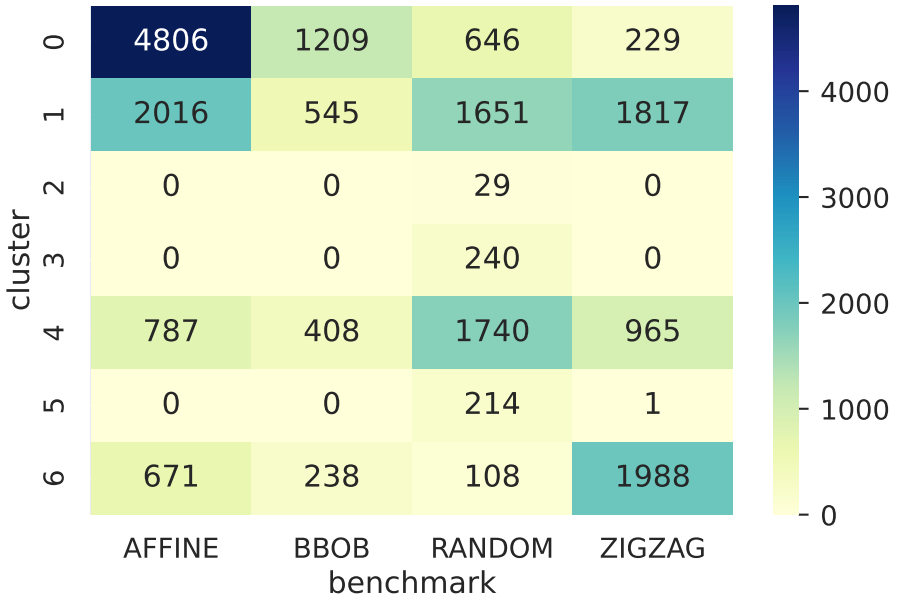



Figure 4.6: Number of problem instances from each benchmark in the clusters generated using ELA features with a sample size of $100d$.

the entire set in the range [0,1], and estimate the optimal number of clusters using the elbow method. Finally, we cluster the problem instances using KMeans clustering with Euclidean distance. Figures 4.6 and 4.7 show the number of problem instances from each benchmark that belong to each of the clusters. Figure 4.6 refers to the results of the clustering performed using ELA features calculated on a $100d$ sample, while figure 4.7 shows the results of the clustering done using the transformer features calculated on a $100d$ sample. The optimal number of clusters is determined to be seven when the clustering is done using the ELA features, and five when using the TransOpt features. The difference in the optimal number of clusters obtained could be owed to the fact that the TransOpt features are trained for the task of classifying BBOB problem instances, and they are thus likely to embed the other benchmarks close to the BBOB problems (i.e., in the same vector space). When the clustering is done using ELA features, we can see that three clusters are almost exclusively populated by RANDOM problem instances, which is consistent with the analysis of the feature distributions, where we saw that the RANDOM benchmark has features with values outside of the distributions of the other benchmarks. On the other hand, the clustering using the TransOpt features still produces one cluster which is dominated by RANDOM problem instances, however, this cluster also contains two BBOB problem instances. To summarize, the results presented in this subsection provide another confirmation that the RANDOM benchmark contains problem instances which are substantially different from other benchmarks and occupy their own part of the problem landscape.

#### 4.1.2.5 Algorithm performance alignment to problem landscape features

In this subsection, we evaluate if the similarity of the problem landscape features is reflected in the raw algorithm performance. To this end, we analyze the similarity of algorithm performance in problem instances which are deemed to be similar according to their problem landscape features.

Figure 4.8 features the algorithm performance metric as defined in Section 4.1.1.3 of the first instance of the first BBOB problem class (sphere) when $d$=3, as well as the scores

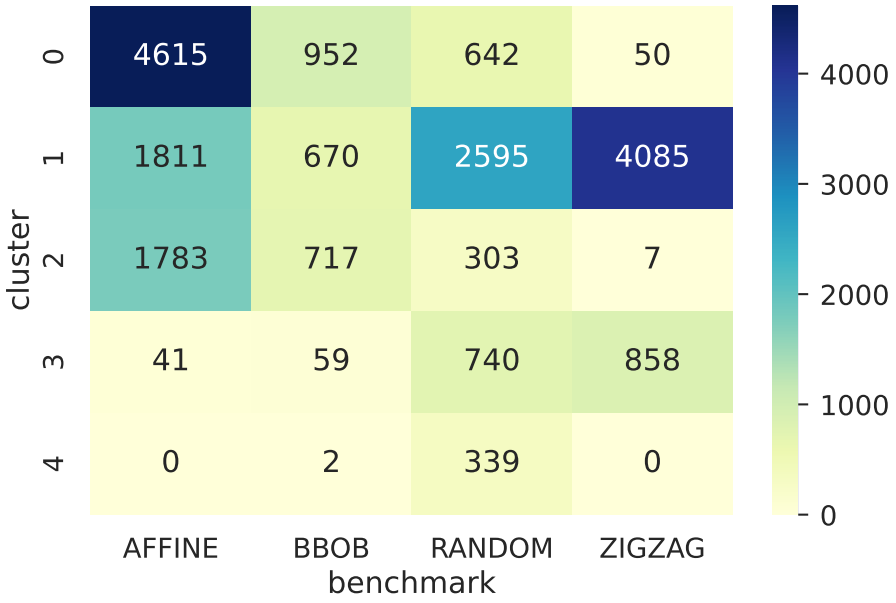



Figure 4.7: Number of problem instances from each benchmark in the clusters generated using TransOpt features with a sample size of $100d$.

of the algorithms on the 10 problem instances from the ZIGZAG benchmark, which are most similar to this problem instance according to the cosine similarity of their problem landscape features. The similarity is calculated separately based on the ELA features, transformer features, and their combination, with a sample size of $100d$. In this case, the algorithm scores are calculated at a budget of 30 iterations for the algorithm execution. In each row, we present a matrix where the rows are different algorithms, the columns are different problem instances, and the values represent the score of the algorithm on the problem instance. The first column represents the query problem instance, i.e. the first instance of the first $3d$ BBOB problem class (Sphere). The remaining 10 columns represent the 10 most similar problem instances from the ZIGZAG benchmark.

The first plot refers to the ELA features, the second subplot refers to the transformer features, and the third subplot refers to the merged ELA and transformer representations.

On the query function, the ES algorithm has the best performance, with the DE algorithm being second-ranked, with a very similar performance. From the figure, we can observe the best-performing algorithm on most of the similar problem instances matches that of the query problem instance. This holds regardless of the type of features used to identify the similar problem instances.

On the other hand, Figure 4.9 presents the opposite case, where similar problem landscape features do not result in similar algorithm performance. In this figure, the query problem instance in the first column is the first instance of the fifth $3d$ BBOB problem class (Linear Slope), where the DE, GA, and PSO provide similar results and are ranked much better than the ES algorithm. Looking at the results for the ELA features in the first plot, we can see that the best-performing algorithm is always ES. This is an example of a very poor alignment of the problem landscape features and algorithm performance. The transformer features and the merged features, provide similar results to the ELA features. These results indicate that there are problem instances for which there is no guarantee that similar problem feature distribution will lead to similar algorithm performance, which raises questions about the predictive power of the evaluated features. Given

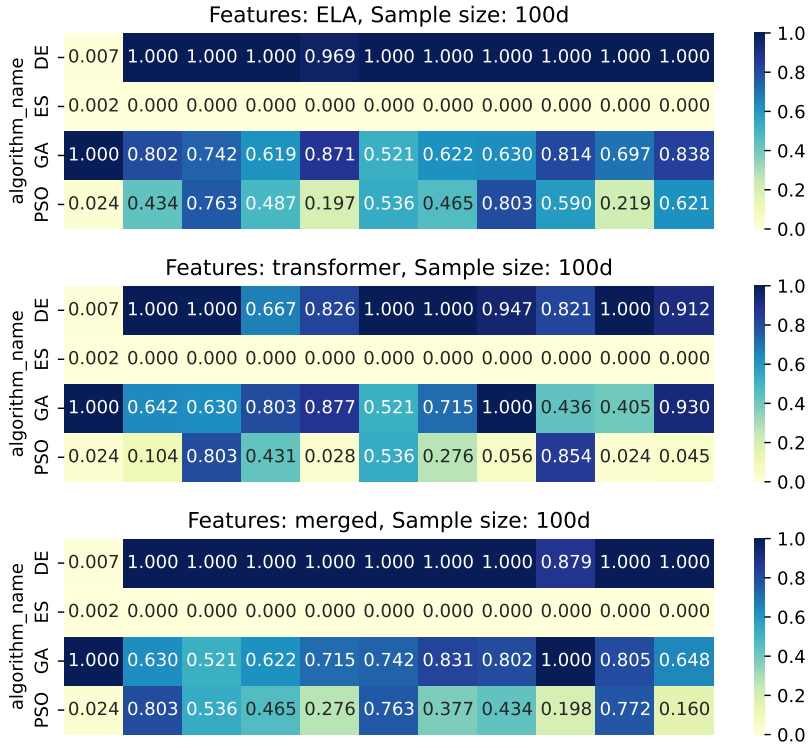



Figure 4.8: Performance alignment of the first instance of the first 3$d$ BBOB problem class and its 10 most similar problem instances from the ZIGZAG benchmark, for a budget of 30 iterations.

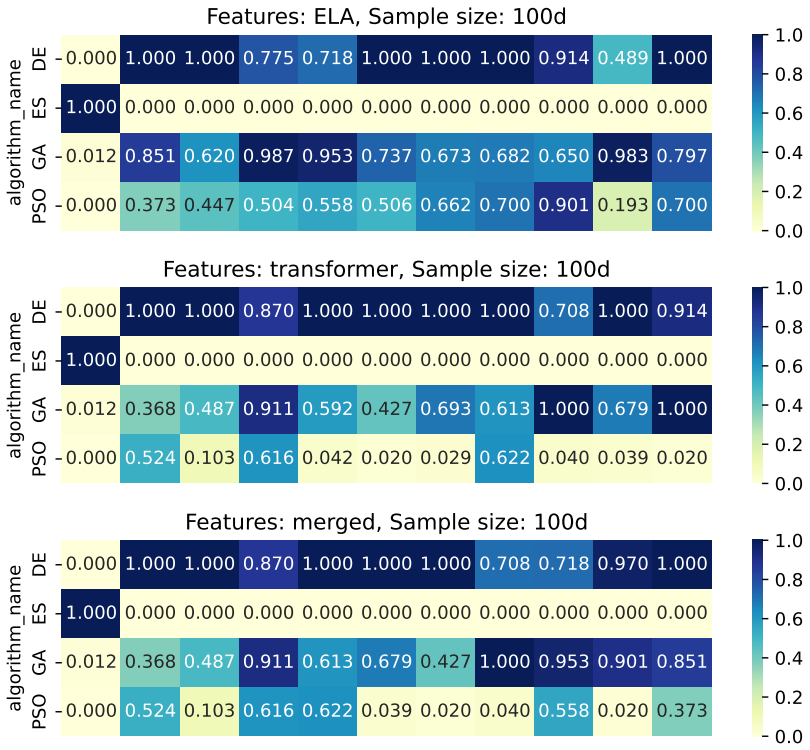



Figure 4.9: Performance alignment of the first instance of the fifth $3d$ BBOB problem class and its 10 most similar problem instances from the ZIGZAG benchmark, for a budget of 30 iterations.

these limitations, trajectory-based features—derived from the actual search behavior of algorithms—represent a promising avenue for capturing aspects of problem-algorithm interactions not reflected in static landscape features.

In order to avoid presenting separate figures for each problem instance and each budget for the optimization algorithm execution, we opt to summarize the information presented in Figure 4.8 in the following manner. We calculate the loss value of the algorithm scores of the query function in the first column and the scores of the top 10 most similar functions in the rest of the columns. We then take the mean of these 10 loss values. In this manner, the performance alignment reflected in each subplot of figures 4.8 and 4.9 is represented by a single number, which indicates how much the algorithm scores of the similar functions differ from the query function.

Figure 4.10a shows the performance alignment of the first instance of the 24 $3d$ BBOB problem classes and their 10 most similar functions from the ZIGZAG benchmark. Each subplot refers to a different budget used for the algorithm execution. Within each subplot, the rows denote the features and sample size used to represent the problem instances when calculating the top 10 most similar functions, while the columns contain the BBOB problem instance used as a query function. We use the BBOB problem instances as the query problem instances since BBOB is the most well-understood and commonly used benchmark. From the figure, we can see that for most problem instances, the similarity in the problem landscape features is reflected in the similarity of the algorithm performance. There are, however, several problem instances where this does not hold. For instance, such is the case with the Buche-Rastrigin Function (denoted as "4_1" in the figure) and the Linear Slope function (denoted as "5_1" in the figure).

To see whether the lack of alignment between the similarity of problem landscape features and algorithm performance leads to worse performance of the AS, Figure 4.10b shows the loss of the AS achieved on the query functions, i.e. the first instance of each BBOB problem class, when the training is done on the ZIGZAG benchmark.

Comparing Figure 4.10a to Figure 4.10b, showing the alignment of algorithm performance to problem instance features, we can see that the AS performs worse on problem instances where there is a lack of alignment between similarity of problem landscape features and algorithm performance. For instance, we can see that the alignment is low on the Linear Slope function, and this is exactly where the loss of the algorithm selector is the lowest. These results show that there are problem instances for which the existing landscape features do not properly capture algorithm performance, which is in line with what was found in a previous work [64]. This indicates that there is a need for further development of problem features which will capture complimentary information related to algorithm performance.

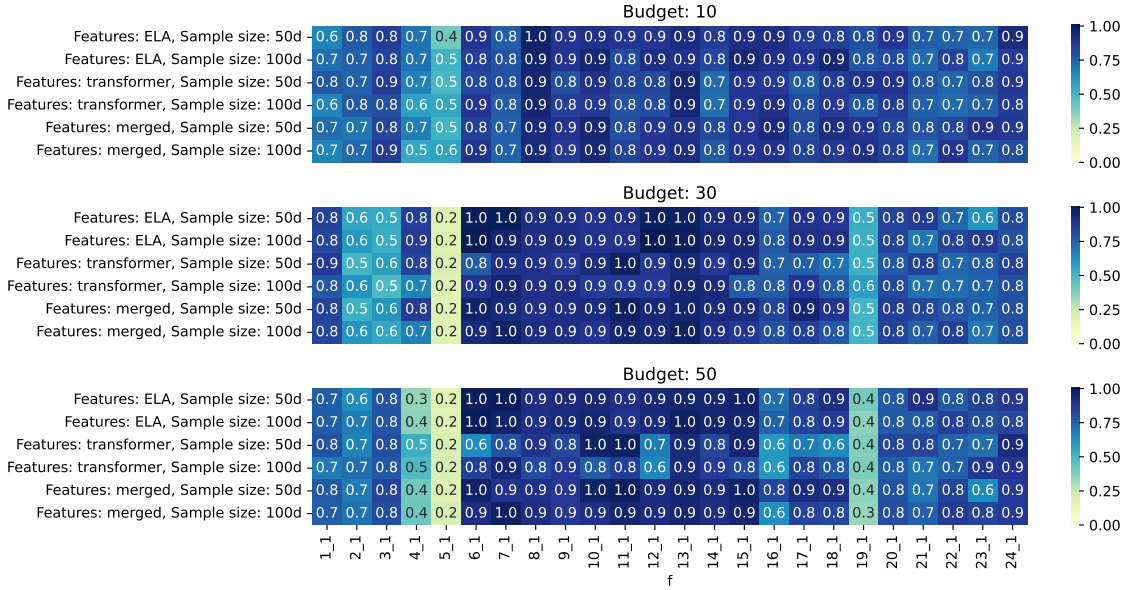
### 4.1.3 Discussion

In this section, we assessed the generalizability of an algorithm selector across different benchmarks. In particular, we considered four benchmark suites (BBOB, AFFINE, ZIGZAG, and RANDOM) across two problem dimensions. We also evaluated whether the recently proposed transformer features can complement the well-established ELA features for the task of algorithm selection. Our results indicate that ELA features provide better results on the BBOB and AFFINE benchmarks, while the transformer features work better on the RANDOM benchmark. Our analysis points out several explanations for this phenomenon.

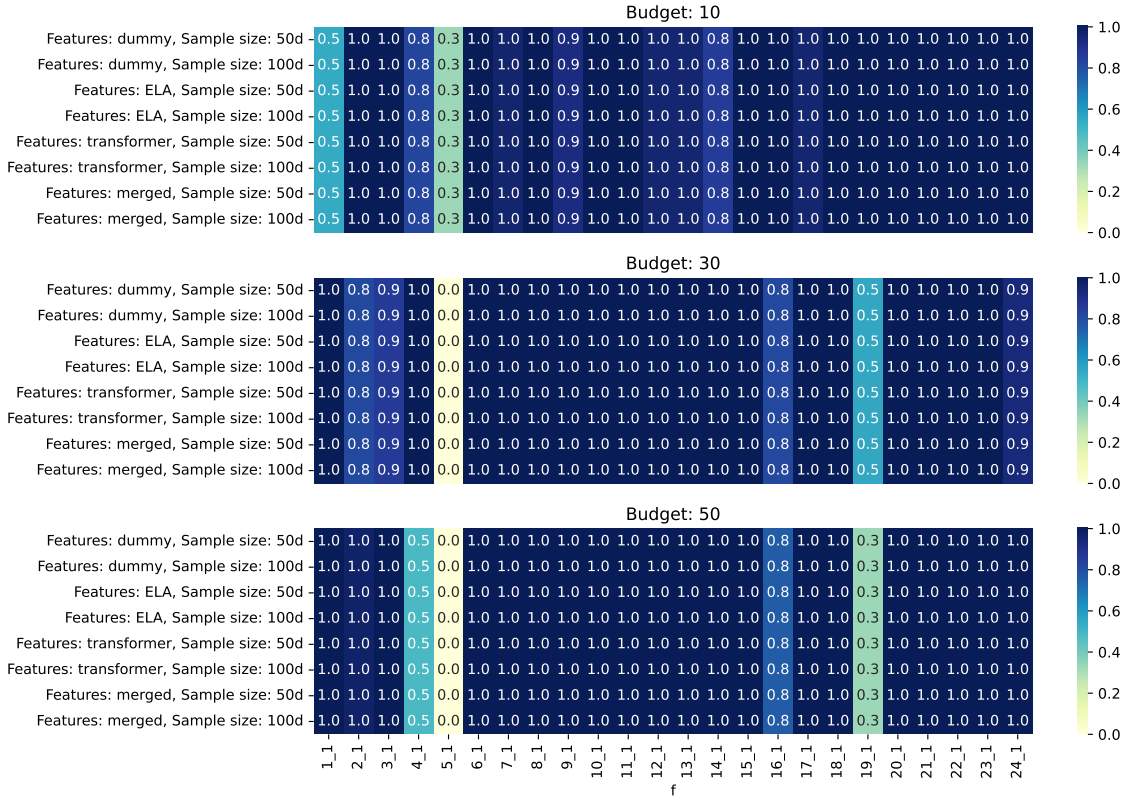
Firstly, when benchmark suites share a similar distribution in the raw algorithm performance across all included algorithms, and they have the same single-best solver, the feature-based AS models exhibit inferior performance. These models, trained on one benchmark suite and tested on others, are outperformed by a baseline model that simply predicts mean performance based on the training data. Next, if the benchmark suite used for training the feature-based AS models differs in raw performance distribution of the algorithms and has a different single-best solver compared to the evaluation suites, these feature-based AS models surpass the baseline dummy model.

We find that the four algorithms (DE, GA, ES, and PSO) are ranked in a mostly consistent manner on the BBOB and AFFINE benchmarks, but rankings differ on the ZIGZAG and RANDOM benchmarks for different problem dimensions. These benchmarks also have similar feature distributions across the problem space. However, the algorithm rankings on the RANDOM benchmark deviate from the rankings on the other benchmarks, and the ELA feature values for some of the problem instances in this benchmark fall outside the ranges observed in the other benchmarks. This implies that the RANDOM benchmark contains some problem instances that are dissimilar to any problem instances from the other benchmarks in terms of ELA features. This is also the case for the transformer features, however, to a smaller extent. The values of the transformer features of the RANDOM benchmark are more consistent with the other benchmarks. A possible reason for this is that the transformer features require the normalization of the objective function values before feature extraction. Further investigation is needed to determine whether normalizing the objective function values before computing ELA features could improve their generalizability across benchmarks.

Our evaluation highlights a common challenge faced by AS models that there are problem instances for which there is no guarantee that similar problem features will lead to similar algorithm performance. This implies that the tested features might lack the neces-




(a) Performance alignment of the first instance of each 3$d$ BBOB problem class to its most similar 10 problem instances from the ZIGZAG benchmark.




(b) Loss achieved by the AS trained on the ZIGZAG benchmark, when tested on the first instance of each of the 3$d$ BBOB problem classes.

Figure 4.10: Performance alignment of the first instance of each of the 3$d$ BBOB problem classes to the ZIGZAG benchmark, and loss achieved on these instances when the model is trained on the ZIGZAG benchmark.

sary discriminatory power to accurately gauge algorithm performance in certain instances. This challenge remains consistent across both ELA and TransOpt features, which brings about the need for novel features to complement existing ones. In particular, trajectory-based features offer a potential path forward by capturing dynamic, algorithm-specific information that static features may overlook.

## 4.2 Landscape Features in Single-Objective Continuous Optimization: Have We Hit a Wall in Algorithm Selection Generalization?

In this section, we aim to provide a comprehensive analysis of recently proposed problem landscape features based on deep learning and topological landscape analysis, and compare them to the most commonly used ELA feature groups. For this purpose, we leverage a benchmark composed of affine combinations of BBOB problem instances to analyze the generalizability of AS models using state-of-the-art problem landscape features. In particular, we focus on the TinyTLA [13], DoE2Vec [11], TransOptAS [124], and the DeepELA [12] features. We selected these feature groups due to their recent introduction (all of them have been developed in the last two years), and the fact that there has been no comparative analysis conducted among all of them thus far. Furthermore, these features do not require additional function evaluations and can be calculated on the same sample of problem solutions, enabling a fair feature comparison.

The main objectives of this study are the following:

- Analyzing the complementarity of the feature groups.
- Analyzing the extent to which each feature group captures algorithm performance.
- Evaluating the generalizability of the AS models obtained with each feature group and combinations of them. The features are evaluated for the AS task on the affine problems [23], using different strategies to split the problem instances into training and evaluation splits, in order to understand the degree of generalizability provided by each feature group.

The remainder of the section is structured as follows. Subsection 4.2.1 describes the data and methodology. Subsection 4.2.2 contains the results, addressing each of our objectives. Subsection 4.2.3 discusses our findings, while Subsection 4.3 concludes the paper.

### 4.2.1 Methodology

In this subsection, we describe the methodology, starting off by introducing the problem and algorithm portfolios, followed by the description of the metric used for algorithm performance evaluation. We then describe the feature calculation procedure, the training of the ML model used for AS, as well as the different settings used for its evaluation.

#### 4.2.1.1 Problem portfolio

We use the set of problem instances from the $10d$ AFFINE benchmark introduced in Subsection 4.1.1.1 that are generated using affine recombinations of the BBOB problems. We use Latin Hypercube Sampling [24] to construct a sample of size $50d$ ($d$ being the problem dimension), on which we evaluate all of the problem instances to obtain their objective function values.

#### 4.2.1.2 Algorithm portfolios

We use four algorithm portfolios, which contain different configurations of the Differential Evolution (DE) [136] and Particle Swarm Optimization (PSO) [36] algorithms. For the DE algorithm, we are randomly initializing the "CR" (crossover constant), "F" (weighting factor), and "variant" (controling the crossover and selection strategies) parameters.

For the PSO algorithm, we are configuring the "initial velocity" parameter (which takes values of "random" when the velocity vector is initialized randomly or "zero" when the particles start to find the direction through the velocity update equation), the "adaptive" parameter which controls whether the inertia, social and coginitive impact are changed dynamically or not, and "w" indicating the inertia value.

The four different algorithm portfolios that we evaluate are as follows:

- The 5DE algorithm portfolio contains five different configurations of the DE algorithm, whose parameter initialization can be found in Table 4.2.
- The 5PSO algorithm portfolio contains five different configurations of the PSO algorithm, whose parameter initialization can be found in Table 4.3.
- The 5DE+5PSO algorithm portfolio contains both the DE and PSO algorithms used in portfolios 5DE and 5PSO.
- The 2DE+2PSO algorithm portfolio contains the two best performing algorithms from each of the 5DE and 5PSO algorithms portfolios

We use multiple algorithm portfolios in order to investigate both the scenarios where all of the algorithms come from the same algorithm class (automatic algorithm configuration), and the scenarios where the algorithms can belong to different algorithm classes (automatic algorithm selection). Using multiple algorithm portfolio also enabels us to analyze the performance of the TransOpt features trained for one algorithm portfolio, and applied on another algorithm portfolio.

Table 4.2: Parameters of the DE configurations.

| | CR | F | selection | crossover |
|---|---|---|---|---|
| DE1 | 0.5 | 0.5 | random | binary |
| DE2 | 0.6 | 0.4 | best | exponential |
| DE3 | 0.9 | 0.7 | best | exponential |
| DE4 | 0.3 | 0.8 | best | binary |
| DE5 | 0.8 | 0.4 | best | binary |

Table 4.3: Parameters of the PSO configurations.

| | initial velocity | adaptive | weight |
|---|---|---|---|
| PSO1 | zero | True | 0.9 |
| PSO2 | zero | False | 0.9 |
| PSO3 | random | True | 0.9 |
| PSO4 | random | False | 0.9 |
| PSO5 | random | False | 0.7 |

#### 4.2.1.3 Algorithm performance metric

We use the metric defined in Subsections 3.2.1.4 and 4.1.1.3 to capture algorithm performance.

#### 4.2.1.4 Feature calculation

In this subsection, we explain the calculation of each of the investigated feature groups.

**ELA:** We compute ELA features from the following feature groups using the *flacco* Python library [70]: *disp*, *ela_distr*, *ela_level*, *ela_meta*, *ic*, *nbc*, and *pca*. In this way, we obtain a total of 62 features for each problem instance. Please note that we selected these features groups since they do not require additional function evaluations (which is the case for other feature groups such as *local_search*, *convexity* and *curvature* features), and can hence be calculated on the same sample of solutions as the other feature groups for a fair comparison. In [82], [142], it has been shown that calculating ELA features on a sample where the objective function values scaled to be in the range [0,1] using min-max scaling can improve their generalizability. The methodology of the other investigated feature groups also includes some type of scaling of the objective function values. For these reasons, we additionally evaluate a version of the ELA features calculated on samples where the objective function values are scaled in the range [0,1] using min-max scaling. We refer to these features as *ela_scaled*.

**TinyTLA:** In the process of topological feature extraction, problem samples are treated as a point cloud from which features are extracted. Both the candidate solutions (x-values) and objective function values (y-values) are first scaled to be between 0 and 1 for each problem instance separately. A *volume* transformation is applied to the x-values as described in [13]. The goal of this transformation is to reduce the features' sensitivity to some problem transformations such as shifting and scaling. The euclidean distances between the x-values and y-values for each pair of samples are calculated independently. The final distances between each pair of samples are obtained by calculating the weighted sum of the distances between the x-values and the y-values. In our case, the weight parameter for combining the distances, $\alpha$ is set to 0.3.

The thereby obtained distance matrix is used to compute persistence diagrams, which encapsulate information about topological structures in the point cloud. Subsequently, numerical features are constructed by transforming the persistence diagram into a persistence image, whose pixels produce a high-dimensional vector that characterizes the problem instance in terms of topological properties. The size of the kernel used to map the persistence diagrams into persistence images is set to 0.002. The selection of all parameter values was based on the analysis of the best-performing parameter combinations we have investigated in [13].

The above feature calculation procedure is repeated for homologies of dimensions 0, 1, and 2 ($H_0$, $H_1$ and $H_2$), which capture different types of topological structures. The 0-dimensional homology $H_0$ describes the connected components of a simplicial complex, the 1-dimensional homology $H_1$ represents 1-dimensional loops (structured like a doughnut), while the 2-dimensional homology $H_2$ captures 2-dimensional cavities. We obtain 50 features from $H_0$ and 2,500 features from each of $H_1$ and $H_2$. The features from the three homologies are concatenated into a single vector, so we obtain a total of 5,050 features from the TinyTLA approach. For more details on this methodology, we refer the reader to the original article [13].

**DoE2Vec:** The training of the DoE2Vec model follows the default implementation in the *doe2vec* python library [147]. The objective function values are scaled in the range [0,1]. The variational autoencoder is trained on a set of 250,000 randomly generated functions for

a total of 100 epochs. The dimension of the latent representation is set to 32, meaning that the model produces a representation of size 32 for each problem instance. We use a custom sample of candidate solutions to evaluate the problems generated by the random function generator, obtain their objective function values, and train the model. The same sample is then used to evaluate the affine problem instances, obtain their objective function values, and pass them through the model to obtain their latent representation used as problem landscape features. This is done so that the sample is fixed across all feature groups to ensure a fair comparison.

**TransOptAS:** The TransOptAS model is trained to predict algorithm performance on a set of 30,000 randomly generated problems. All algorithms are executed on these functions to obtain the performance data. The model includes two layers in the encoder of the transformer architecture, each one consisting of two heads. The size of the embedding is set to 50, meaning that the generated problem representations are of size 50. The model is trained for 100 epochs using an early stopping strategy which terminates the training early if an improvement of 0.001 in the validation loss is observed for 3 epochs. Once the model is trained, the weights are frozen and the regression layer (which predicts algorithm performance) is removed from the TransOptAS model. The next-to-last layer is then used for the purpose of generating problem representations. The objective function values of the problem samples are scaled in the range [0,1] and are then passed through the model to produce their representations. A separate model is trained for predicting the performance of each algorithm portfolio, resulting in the training of four models and thereby four variations of the TransOptAS features generated for each problem instance. We name these features according the the algorithm portfolio used for the model training, i.e., *transoptas_ 5DE*, *transoptas_ 5PSO*, *transoptas_ 5DE+5PSO* and *transoptas_ 2DE+2PSO*.

**DeepELA:** As DeepELA is already pre-trained, no further training or fine-tuning was done. Instead, the trained *Large_ 50d* model (as proposed in [12]) was applied to the same samples of candidate solutions used for the calculation of the other features. The predicted features are guaranteed to be invariant to the order of the candidates but they are not guaranteed to be in variant to the order of the dimensions (see [98]). Hence, we compare a single forward pass to ten repeating forward passes as outlined in [98]. The authors introduced the latter approach to improve the robustness of the generated features but reported only a slight improvement. Specifically, instead of performing a single forward pass to compute the feature vectors, this approach involves computing ten individual forward passes with different random permutations of the decision space to compensate for small fluctuations of the feature vectors. Afterwards, the arithmetic mean of the ten feature vectors is used as a representation. We will refer to the two approaches *deep_ ela_ large_ r1* (calculated with a single forward pass) and *deep_ ela_ large_ r10* (calculated as the mean of ten forward passes).

#### 4.2.1.5 Automated algorithm performance prediction model

We use a RF model to perform multi-target regression, where the model predicts the performance score of each algorithm as defined in Subsection 3.2.1.4.

We address the task as a multi-target regression for the following reasons:

- Unlike multi-class classification, which predicts a single best algorithm (as done in [8]), this method captures the performance of all algorithms. This approach avoids penalizing a model for selecting one of two algorithms with very similar performance.

- In contrast to predicting algorithm ranking (as done in [9]), where each algorithm is assigned an integer score based on its performance, this approach captures fine-

grained differences in performance, allowing algorithms with similar best-obtained values to receive comparable scores.

We chose the RF model because it has good performance on tabular data [140], it is robust against differently scaled feature ranges, and it provides feature importance values that help with interpretability. The RF model is executed using the default configuration parameters in the *scikit-learn* [146] Python library version 1.2.2, meaning that the number of estimators (*n_estimators* parameter) is 100, the criterion for measuring the quality of a split (*criterion* parameter) is the gini impurity, minimum number of samples required to split an internal node (*min_samples_split* parameter) is set to 2, the minimum number of samples required to be at a leaf node (*min_samples_leaf* parameter) is set to 1, the minimum node impurity threshold required for splitting each node (*min_impurity_decrease* parameter) is set to 0, the number of features to consider when looking for the best split (*max_features* parameter) is set to the square root of the total number of features, while the rest of the parameters are set to None.

We do not perform parameter tuning in order to evaluate the impact of the proposed features only, on a fixed model configuration. We train a separate RF model for each algorithm portfolio and for each feature representation. Apart from considering the individual use of a single feature group, we also consider the representations obtained by concatenating pairs of feature groups, as well as concatenating all feature groups. We compare the performance of the RF models to a simple baseline "dummy" model which predicts the mean performance of the algorithms on the train set.

#### 4.2.1.6 Algorithm selector performance metric

We use the Loss metric defined in Section 3.2.1.6 to evaluate the performance of the AS model.

#### 4.2.1.7 Evaluation settings

To account for the different AS evaluation approaches present in the literature and provide a comprehensive, all-encompassing view of the contribution of the different feature groups, we explore several evaluation settings, differing in how the problem instances are split into training and testing sets. The evaluation settings are listed in increasing level of difficulty.

**Instance Split:** This evaluation setting corresponds to the commonly used LIO evaluation on the BBOB benchmark suite, meaning that one instance of each generated affine problems is in the test set, while the remaining four instances are in the training set. This is the most lenient evaluation setting, since it is guaranteed that for each testing instance, the training set contains problem instances which are derived from the same pair of parent problem classes used for generating the test instance. For a specific train/test fold, the first problem instance created by combining pairs of problem classes would be placed in the test set, while the remaining instances (2-5) would be placed in the training set.

**Random Split:** In the random split evaluation setting, the affine problems are split into train and test set randomly, disregarding any information about the problem class or instance identifier. To provide an example, let us assume we have constructed a single train/test fold from all the affine functions. If we focus exclusively on affine pairs from problem classes 1 and 2, with five problem instances, in some folds, all five instances might appear in both the train and test sets, while in other cases, all 5 might be assigned either to the train or the test set. In other words, there is no guarantee that at least one instance of a specific pair will be in the train or test set.

**Problem Combination Split:** In this evaluation setting, the testing set contains all instances generated by combining one BBOB problem class with all of the remaining 23

BBOB problem classes. However, the training set does not contain any instances derived from the first problem class, and instead contains only combinations of the remaining BBOB problem classes. This means that the training set contains instances which are derived from one of the parent classes in the test set, but not the other one. For example, in one fold, in the test set we could have all problem instances which are generated by combining instances of problem class 1 with all other 23 problem classes. In this case, the training set would contain only instances obtained by combining problem classes 2-24, but would not contain any instances which use problem class 1.

**Problem Split:** In this evaluation setting, the testing set contains combinations of two problem classes, while the training set contains the combinations of the remaining 22 problem classes. This is the most difficult evaluation setting, where the problem instances in the test set are guaranteed not to be derived from any problem class used in the generation of the training set. In this case, the test set in one fold could contain all problem instances generated by combining problem class 1 only with problem class 2, while the test set would not contain any problem instance generated by combining problem class 1 with any other problem class, nor problem class 2 with any other problem class.

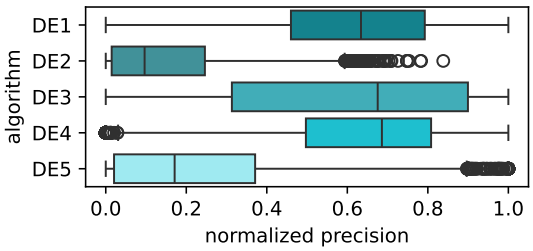
### 4.2.2 Results

In this section, we present the findings of the analysis of the feature groups. Our results are structured as follows:

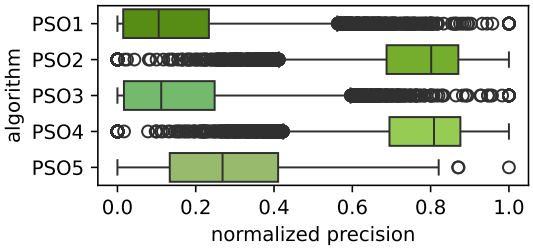
- Subsection 4.2.2.1 provides an overview of the performance of the four algorithm portfolios.
- Subsection 4.2.2.2 analyzes the complementarity of the investigated feature groups
- Subsection 4.2.2.3 investigates the degree to which the feature representations capture algorithm performance.
- Subsection 4.2.2.4 presents the results of the AS models using different feature types and evaluation settings.

#### 4.2.2.1 Algorithm performance

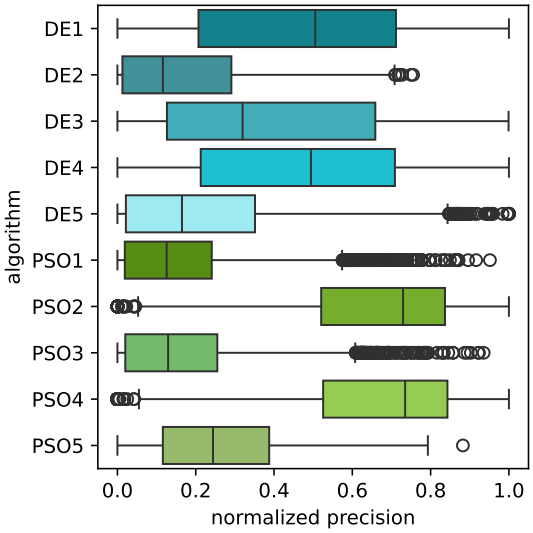
Figure 4.11 shows the distribution of the normalized precision scores (defined in Subsection 3.2.1.6) achieved by each algorithm in the four algorithm portfolios. In the 5DE portfolio presented in Subfigure 4.11a, we can see that the DE2 and DE5 algorithms provide the best results for the majority of the functions, with a median normalized precision of 0.09, and 0.17, respectively. On the other hand, the DE1, DE3 and DE4 provide median normalized precision in the range 0.63-0.68. Looking at the 5PSO portfolio in Subfigure 4.11b, we can see that the PSO1 and PSO3 algorithms, where the inertia and cognitive and social impact are changing dynamically over time, provide the best results. In Subfigure 4.11c, presenting the 5DE+5PSO portfolio, we can see that the worst performing algorithms are PSO2 and PSO4 with a median normalized precision of around 0.72, followed by the DE1 and DE4 algorithms with a median normalized precision of approximately 0.50. The DE2, PSO1 and PSO3 have a similar median performance of around 0.13, closely followed by DE5. Combining these algorithms in the 2DE+2PSO portfolio gives a more complementary algorithm portfolio, where the median performance of the best-performing PSO1 algorithm is around 0.33.



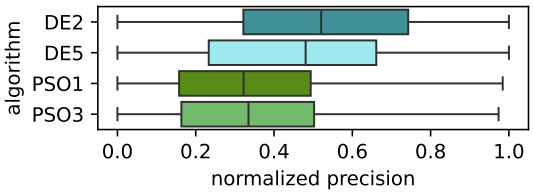

(a) 5DE portfolio




(b) 5PSO portfolio




(c) 5DE+5PSO portfolio




(d) 2DE+2PSO portfolio

Figure 4.11: Distribution of the normalized precision scores in each algorithm portfolio. A lower normalized precision signifies a better performance of an algorithm inside a portfolio.

#### 4.2.2.2 Feature complementarity

Here, we investigate the complementarity of the feature groups. Figure 4.12 shows a clustermap of the Spearman correlations of individual pairs of features. The Spearman correlation is calculated for each pair of features, by taking their values across all problem instances created using affine recombinations. The labels on the x and y axis indicate the names of the features being compared, while the heatmap indicates their Spearman correlation. The dendrogram is based on a hierarchical clustering performed on top of the Spearman correlations, in order to group features with similar correlations together. Due to the high number of features, it is not possible for us to show all of the feature names in a readable way. For this reason, the feature group is additionally indicated by a color bar on the top and left side of the heatmap.

For improved visibility, we exclude some of the feature groups from this figure. In particular, we exclude the *ela* features in favor of *ela_scaled*, since a lot of them are not sensitive to the scaling of objective function values and are therefore identical to the *ela_scaled* features. Similarly, we exclude the *deep_ela_large_r1* features since they are highly similar to the *deep_ela_large_r10* features. From the four investigated *transoptas* variants, we include only *transoptas_5DE* and *transoptas_5PSO* since they are trained on algorithm portfolios consisting of completely different algorithm classes (DE and PSO), and it is of our interest to analyze whether similar representations are learned in this case. We remove the features from each feature group which have constant values for all problem instances. Since the *tinytla* feature group contains a total of 5,050 features, which is a prohibitively high number for visualization, we reduced the number of features. In particular, we performed Principal Component Analysis (PCA) dimensionality reduction to reduce the features of each homology to 20 dimensions, which preserves 99% of the explained variance. Please note that this transformation of the *tinytla* features is only done for visualization purposes for Figures 4.12 and 4.13, and the features are used in their original form in the remaining analyses.

Along the colorbar, there are several large clusters composed entirely of *tinytla* features, a few clusters containing only *ela_scaled* features, and a few clusters consisting of the two variants of *transoptas* features. The *transoptas_5DE* and *transoptas_5PSO* are commonly clustered together, indicating that even though the features were learned from portfolios of different algorithm classes, they capture similar information. However, the clusters they produce have varying correlation levels, meaning that there are subgroups within these features that capture diverse information.

The *deep_ela_large_r10* and *doe2vec* features occasionally produce smaller homogeneous clusters (with features coming from the same group), but are generally placed in diverse clusters. Looking at the three largest clusters of *ela_scaled* features, we can see that the biggest cluster is composed of the *disp* (dispersion) subgroup. The features within this cluster also have a high positive correlation with each other. The second largest cluster contains the features capturing the runtime costs from different subgroups, while the third largest one contains features from the *ela_level* subgroup.

Figure 4.13 focuses on the comparison of the previously investigated feature groups to the *ela_scaled* features. Here, the *ela_scaled* features are featured on the vertical axis, while the remaining feature groups are on the horizontal axis. The goal of this analysis is to gain some intuition about the information captured by the deep-learning based features, which are not interpretable, by relating them to some of the *ela_scaled* features. From this figure, we can see that only a few of the *tinytla* features have strong correlations with the *ela_scaled* features, while the majority of them are not correlated. However, here it is important to stress that these features have undergone a PCA dimensionality reduction to reduce the number of features for visualization purposes. Looking at the *doe2vec* features,

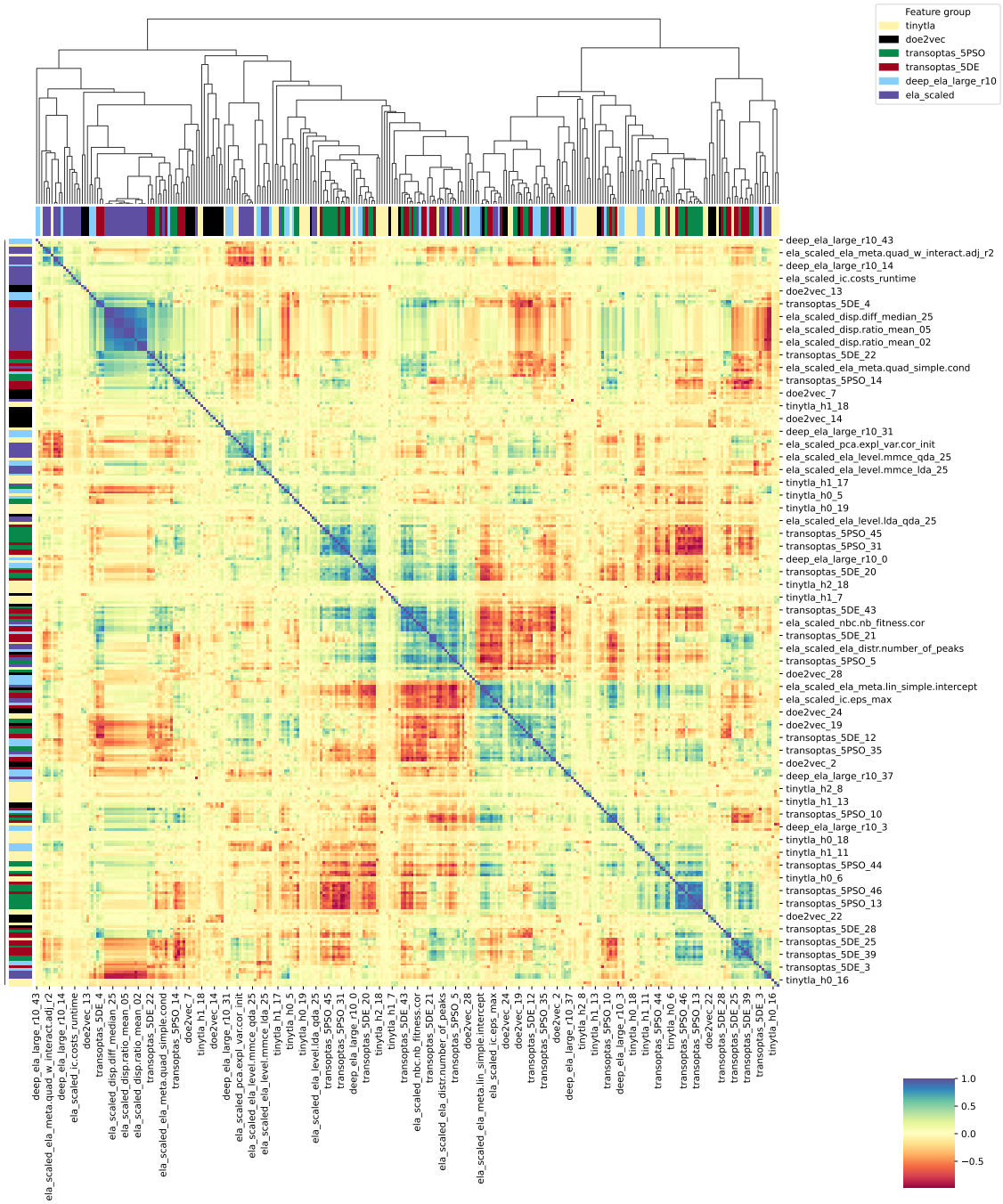



Figure 4.12: Spearman correlation clustermap between pairs of features from different feature groups.




Figure 4.13: Spearman correlation clustermap between the *ela_scaled* features and the other feature groups.

we can see that most features are not correlated with the *ela_scaled* features. Individual *doe2vec* features have strong correlations with the *ela_scaled_nbc* subgroup and all of the features in the *ela_scaled_disp* subgroup. On the other hand, for the *deep_ela* we observe stronger correlations with different subgroups. The *transoptas* variants exhibit strong correlations with some specific features. In particular, a lot of the features are strongly correlated with the *ela_scaled_distr* subgroup, capturing properties of the distribution of objective function values, as well as the *ela_scaled_ic* subgroup, containing the information content features, and the *ela_scaled_nbc.nb_fitness.cor* feature.

To further understand the complimentarity of different feature groups, our next experiment focuses on evaluating the consistency of similarity of problem pairs produced by different feature groups. The goal of this experiment is to identify feature groups where there is "disagreement" about whether two problems are similar to each other or not. Pairs of features producing different similarities for the same pair of problems can be considered an indicator that these features capture diverging views of the problem instance and may be complimentary.

To this end, we randomly sample 10,000 pairs of problems and calculate the cosine similarities between their representations produced with each feature group. Before cosine similarity is calculated, constant features are removed and the remaining features are scaled to be in the range [0,1] using min-max scaling. Figure 4.14 shows the cosine similarity produced by each combination of features. The pair of problems are plotted in the form of a scatterplot, where the two coordinates denote the cosine similarity between these problems, obtained with two different feature representations.

The first feature name in the title of each subplot refers to the horizontal axis, while the second feature name refers to the vertical axis. For enhanced visibility, a histogram denoting the distribution of values is plotted on top of the scatterplot, where brighter values indicate higher values. In the case of perfect consistency in the similarities produced by two feature groups, the subplot would resemble a diagonal bottom-up line. We can immediately see that this is the case for the *deep_ela_large_r1* and *deep_ela_large_r10* features, as well as the *ela* and *ela_scaled* features. When problems are represented in terms of the *ela* and *ela_scaled* features, the least similar problem pairs have similarities around 0.50. On the other hand, in terms of the *tinytla* features, the lowest similarities reach as low as 0.25. Looking at the comparison of the *tinytla* features to other feature groups, we can see that there are some problem pairs where *tinytla* features produce similarities as low as 0.25, whereas other feature groups produce similarities above 0.80. This indicates that *tinytla* features capture information which is sometimes different than the other feature groups.

Focusing on the comparison of the *transoptas* features with other *transoptas* features, we can see that the shape of the distributions produced is quite similar. The cosine similarity of most of the problem pairs exceeds 0.80, with the least similar problem pairs having similarities as low as 0.70.

In summary, when comparing feature sets, we observe that in most cases there is some agreement between feature groups when problem instances are extremely similar. When one group assigns high similarity ($> 0.95$), it usually means that the other feature groups will also assign high similarity. However, when pairs of instances are not as similar, one feature set might assign high similarity while another might assign low similarity, indicating a disagreement between feature sets on how similar the pairs of problem instances are.

#### 4.2.2.3 Algorithm performance alignment to problem landscape features

In order for a ML model to work, the input features should be able to capture the target variable. To understand to what extent this is true for each of the feature types, in this section, we analyze the alignment of each feature type to algorithm performance.

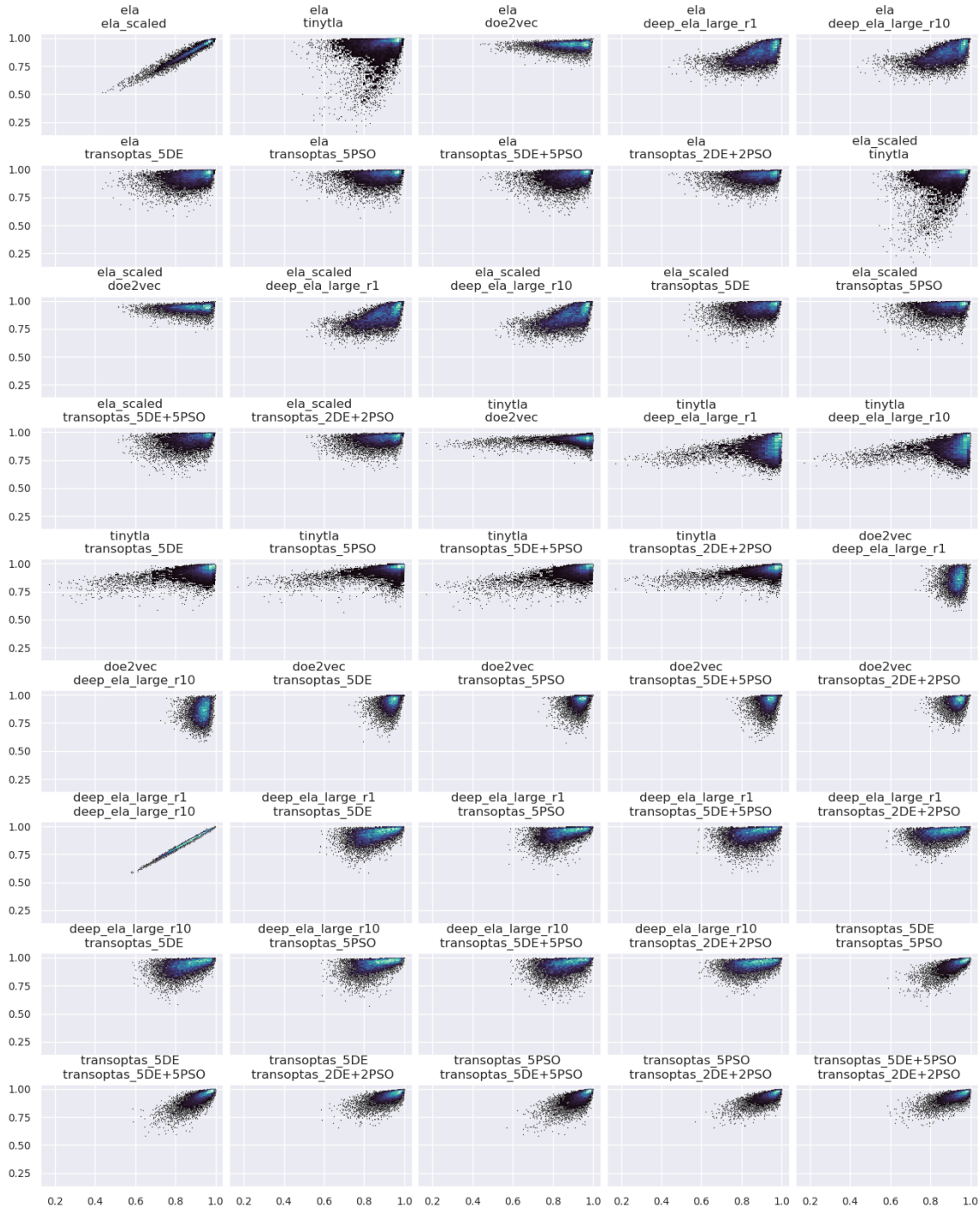



Figure 4.14: Cosine similarity of problem pairs, calculated with different feature sets. The first feature name in the title of each subplot refers to the horizontal axis, while the second feature name refers to the vertical axis.

As a first experiment, we investigate whether problems with similar feature representations exhibit similar algorithm performance. More specifically, we sample 1,000 of the affine problems and measure the cosine similarity of their feature representations and the cosine similarity of their performance vectors, containing the score of each of the algorithms in the algorithm portfolio. Figure 4.15 shows the results for each feature type, for the 2DE+2PSO portfolio. We chose this portfolio for illustration purposes, since it exhibits the highest algorithm complementarity. Each subplot shows the results for a different feature type. The horizontal axis shows the similarity in feature space, while the vertical axis shows the similarity in performance space. Both axis are accompanied by a histogram representation of the marginal distribution. The red line represents the mean value of the performance similarity for a given feature similarity (rounded to two decimal points), while the black line denotes the median. The main plot area displays hexagonal bins, where the color intensity of each hexagon indicates the number of data points within that bin. Darker hexagons represent areas with a higher concentration of data points. Please note that the number of points within each hexagon has undergone a logarithmic transformation before being mapped to its corresponding color, to prevent from hexagons having a low number of points being shown as empty. In an ideal case, the feature similarity would be equal to the performance similarity, i.e., most of the points would be located on the diagonal from the top right to the bottom left corner. However, we can see that this is not the case for any feature type.

From the marginal distributions, we can see that the feature similarities for all feature types are almost entirely above 0.60, with peaks in the distributions around 0.80-0.90. The *ela* and *ela_scaled* features have a median similarity of 0.89, the *tinytla*, *doe2vec*, and all of the *transoptas* features have median similarities around 0.94, while the *deep_ela* features have median similarities around 0.85. Looking at the red and black lines, denoting the mean and median performance similarity for a given feature similarity, we can see that performance similarity is not usually increasing with the increase of feature similarity. We can only observe an increase when the values of the feature similarity exceed 0.98. In the areas with low feature similarity, we can also observe a zig-zag pattern appearing at the beginning of each line, which can be attributed to a low number of points with high variability in performance similarity located in this area.

From this figure, we conclude that none of the feature groups exhibit a perfect alignment with algorithm performance. The fact that alignment between similarity in features and similarity in performance is only clearly observed when feature similarity is extremely high could indicate that an AS model could correctly infer performance if it has previously observed a similar or almost identical problem instance.

One weakness of the previous experiment is that when measuring the similarity of problem instances using all of the features within one feature group, there may be some features that capture algorithm performance, and others that do not. As a consequence, in the event when a majority of features within a single group are not capturing performance, the results of the previous experiment may be too pessimistic. For that reason, we conduct another experiment which compares the correlation of algorithm performance with individual features within a feature group.

To this end, we calculate the Spearman correlation coefficient between each feature, and the performance (normalized precision) of individual algorithms. Figure 4.16 shows the distribution of Spearman correlation coefficient for features within the feature groups. Each column denotes a single algorithm from the 2DE+2PSO portfolio, while each row denotes one of the different feature groups. In an ideal case, the distribution within each subplot should be bimodal with peaks along the leftmost or rightmost edge, meaning that most of the features should be correlated (positively or negatively) with algorithm

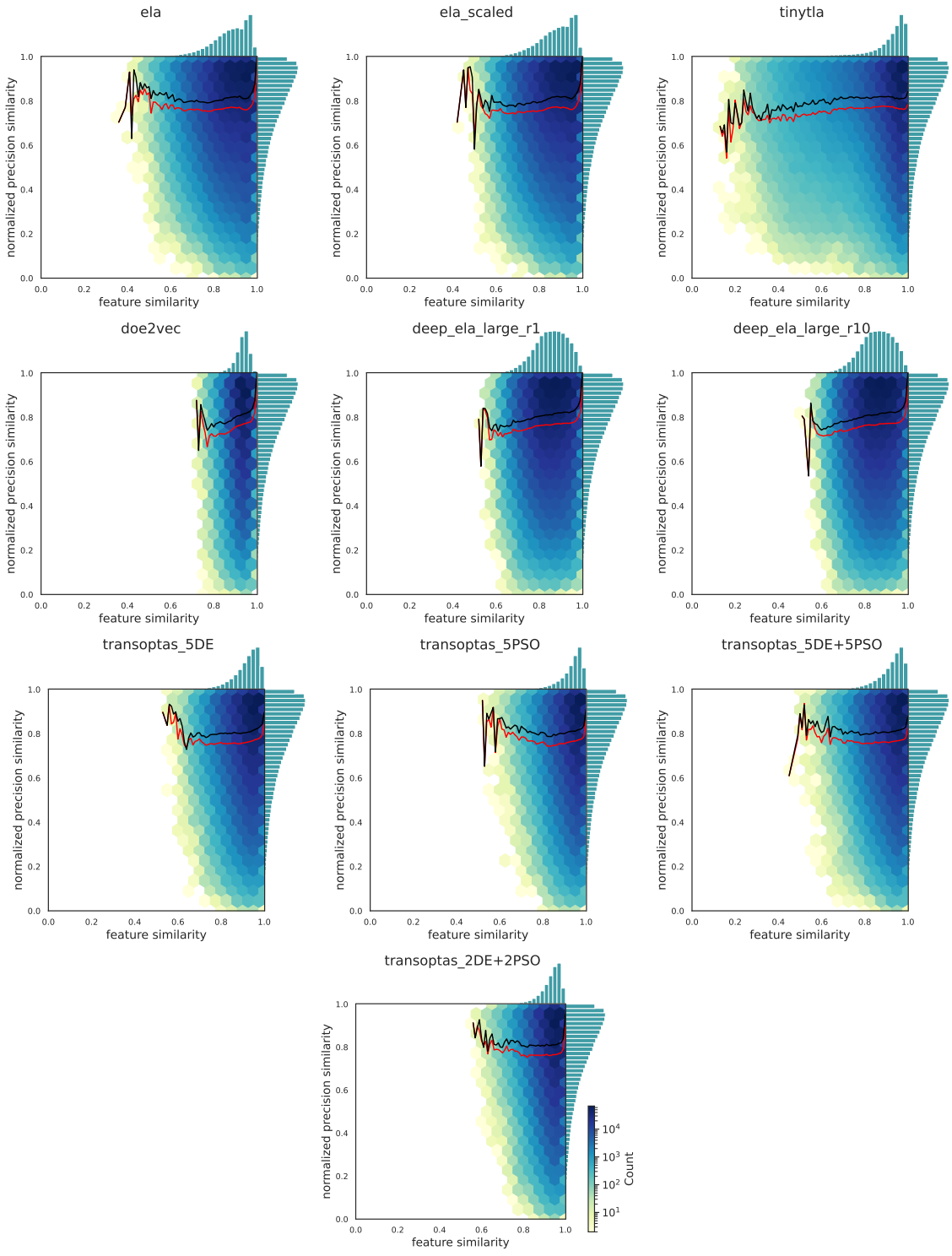



Figure 4.15: Distribution of feature and performance similarities for the 2DE+2PSO portfolio for 1,000,000 problem pairs.

performance. A peak at the center (zero) indicates that a lot of features within a feature group are not correlated with performance. In this case, we do not remove the constant features, since we want to consider how many of the features produced from each feature group are actually useful for capturing performance. As a first observation, we can see that all correlations are between -0.30 and 0.30, indicating that none of the features individually have a very high correlation with algorithm performance. We can see that the distributions of the *ela* and *ela_scaled* features look relatively similar, which is unsurprising since a lot of the features are not sensitive to the scaling of objective function values and are therefore equivalent in both groups. Looking at the *tinytla* features, we can see that the disribution has a large peak at zero. This is due to the fact that a large portion of the *tinytla* have constant values, and their Spearman correlation is therefore set to zero. The *doe2vec* features are are also centered at the zero point, with the most highly correlated feature being around -0.20 and 0.20. The *deep_ela_large_r1* and *deep_ela_large_r10* have nearly identical distributions, with a slightly higher spread than the *doe2vec* features. Interestingly, for the DE5 algorithm, there is a larger number of *deep_ela* features with a negative correlation. Considering the four *transoptas* variants, we can observe that in several cases, the distributions are bimodal and not centered at zero. This is the case for the *transoptas_5DE* and *transoptas_5PSO* features for most of the algorithms.

In conclusion, this analysis shows that none of the investigated feature groups are perfectly capturing algorithm performance, and all of the feature groups have individual features which are not at all correlated with algorithm performance.

#### 4.2.2.4 Algorithm selection results

In this subsection, we present the results from the AS in terms of the loss metric defined in Subsection 3.2.1.6. Figures 4.17, 4.18, 4.19 and 4.20 present the results from the four different evaluation settings: Instance split, Random split, Problem Combination split, and Problem split. Each figure is divided into four subfigures, containing the result for a different algorithm portfolio. The horizontal axis contains the names of the features used to train the model, while the vertical axis contains the value of the loss metric, indicating the performance of the model using the particular set of features for the AS task. Please recall that the loss is supposed to be maximized. In each subplot, there are three horizontal and vertical lines added with the purpose of increasing readability. The three vertical lines are meant to separate the features into groups based on the number of feature groups being combined. In the first group, each feature group is used individually. In the second group, pairs of feature groups are being combined. The third group consists of all of the feature groups being combined together, while the fourth group is the baseline dummy model. The red dashed horizontal line denotes the median AS performance of the original *ela* features, the green dash-dotted line denotes the median AS performance of the *ela_scaled* features, while the black dotted line denotes the median AS performance of the baseline dummy model. These lines are meant to provide an easier comparison of the results obtained by different feature groups to the results of the dummy model and the *ela* features.

Taking into consideration the results from the instance split evaluation in Figure 4.17, we can see that all of the feature-based models generally outperform the dummy model. The difference between the median AS performance of the *ela* model and the dummy model amounts to 0.041, 0.067, 0.032, and 0.011 for the 5DE+5PSO, 2DE+2PSO, 5DE and 5PSO algorithm portfolios, respectively.

The *ela_scaled* features underperform the *ela* features except for the 5PSO portfolio, where they outperform *ela* by a small margin. This means that scaling the objective function values before feature calculation is generally not beneficial in this case. Comparing the results in the first horizontal group, where individual feature groups are used, we can

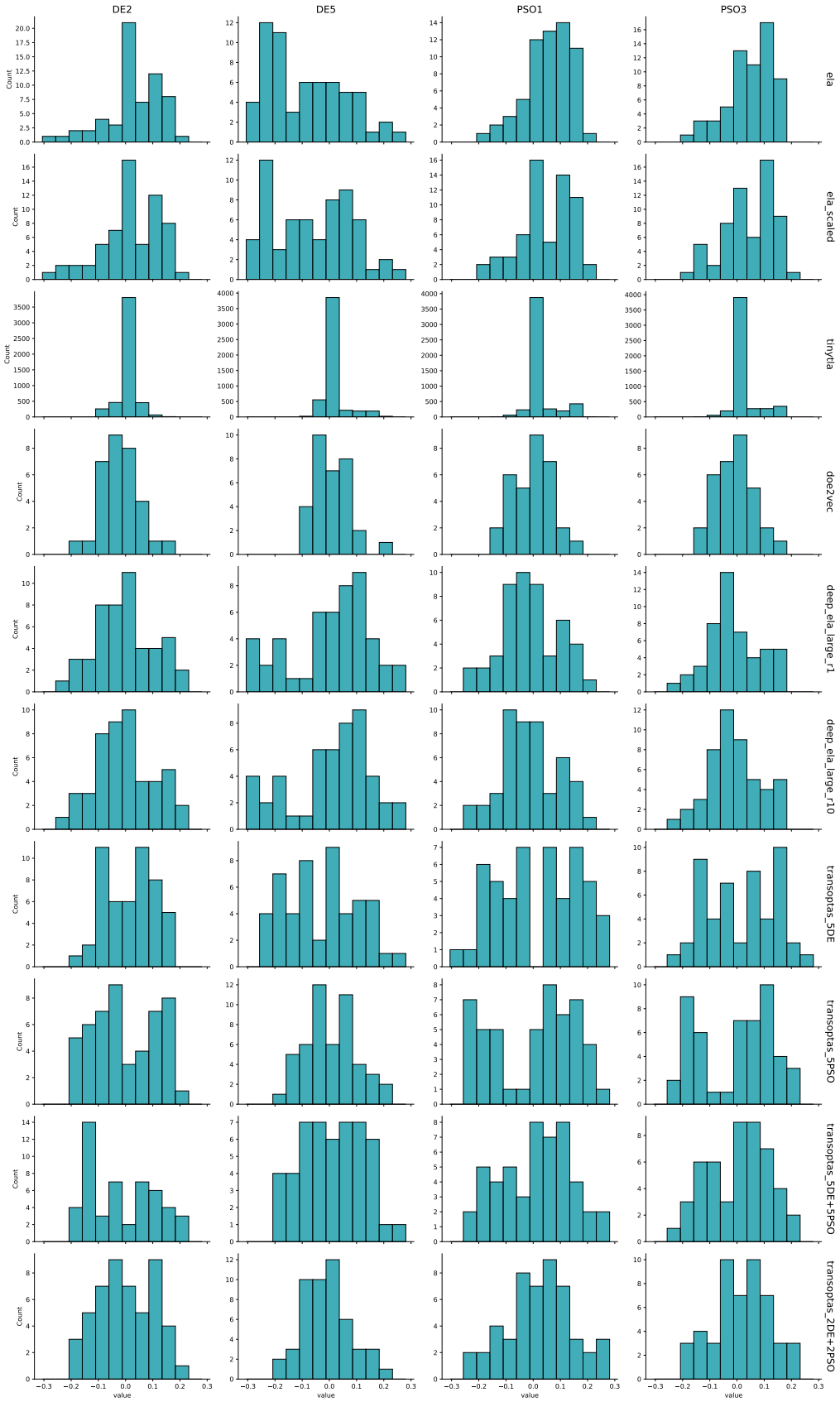



Figure 4.16: Distribution of Spearman correlation coefficient of individual features within feature groups with performance of individual algorithms within the 2DE+2PSO algorithm portfolio.

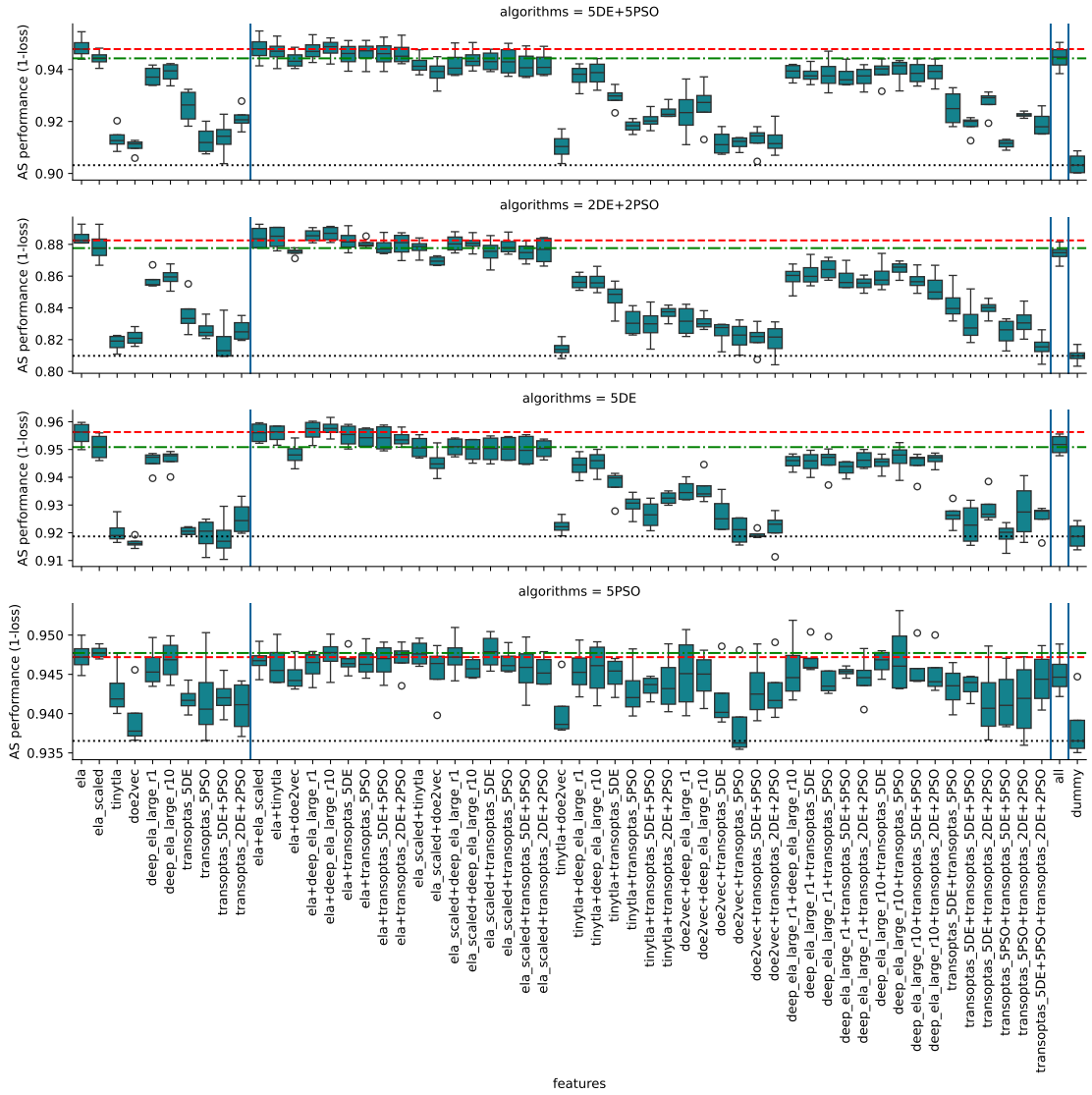



Figure 4.17: AS performance obtained for the different feature groups and algorithm portfolios in the instance split evaluation.

see that the *ela* and *ela_scaled* features are providing the best results, with the two variants of the *deep_ela* features being the second ranked, the *transoptas* variants ranking third, followed by *tinytla* and *doe2vec*.

Looking into the second horizontal group, where pairs of feature groups are combined together, we observe that most of the feature pairs still do not manage to outperform the individual use of the *ela* features. In some rare cases, we do observe a very minor improvement in performance. This is the case when combining the *ela* with the *deep_ela* variants for the 2DE+2PSO portfolio and 5DE portfolio, although the difference is negligible.

Using all of the features together results in no improvement over the individual use of the *ela* features.

Looking at Figure 4.18, showing the results for the random split, we observe a similar pattern of the feature-based models outperforming the dummy model, and the individual use of the *ela* features providing the best results overall with negligible improvements contributed to the other feature groups. The *ela* model outperforms the dummy model by

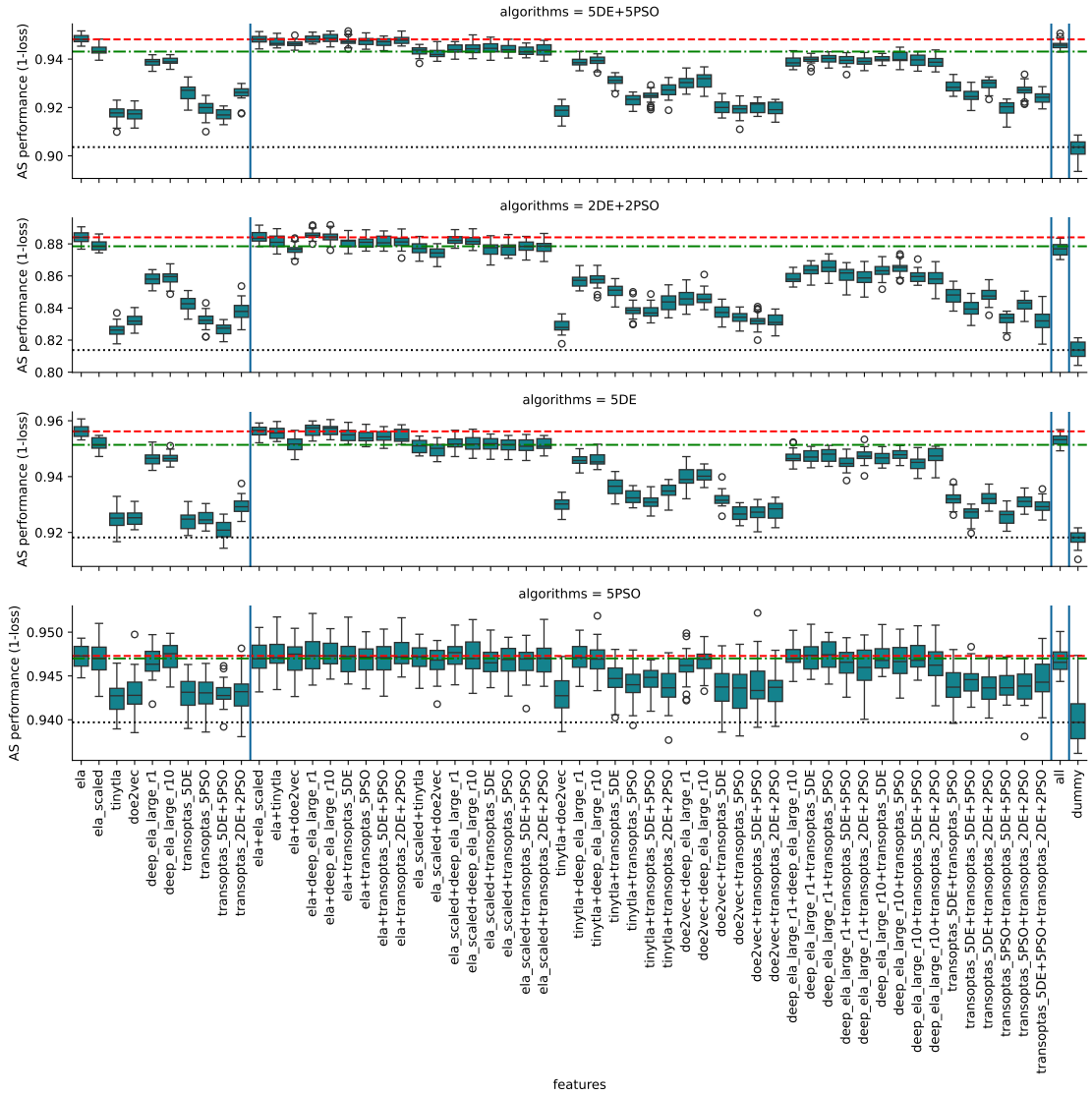



Figure 4.18: AS performance obtained for the different feature groups and algorithm portfolios in the random split evaluation.

0.039, 0.064, 0.033, and 0.007 for the 5DE+5PSO, 2DE+2PSO, 5DE and 5PSO algorithm portfolios, respectively.

Focusing on Figure 4.19, presenting the results from the problem combination split evaluation, where all problems derived from one parent problem class are used for testing, we can see that there is a drop in the improvement provided by the feature-based models compared to the dummy. In this case, difference between the median AS performance of the *ela* model and the dummy model corresponds to 0.032, 0.030, 0.013, and 0.001 for the 5DE+5PSO, 2DE+2PSO, 5DE and 5PSO algorithm portfolios, respectively. For the 5PSO algorithm portfolio, the feature-based models no longer outperform the dummy, or the difference in their performance is close to zero. We can also observe that there are folds for which the performance of all models is drastically lower than the others, which is likely an indicator that there is one problem class which when combined with remaining 23 problem classes, produces problems which are outside of the distribution of the other problems generated by combining the 23 problem classes.

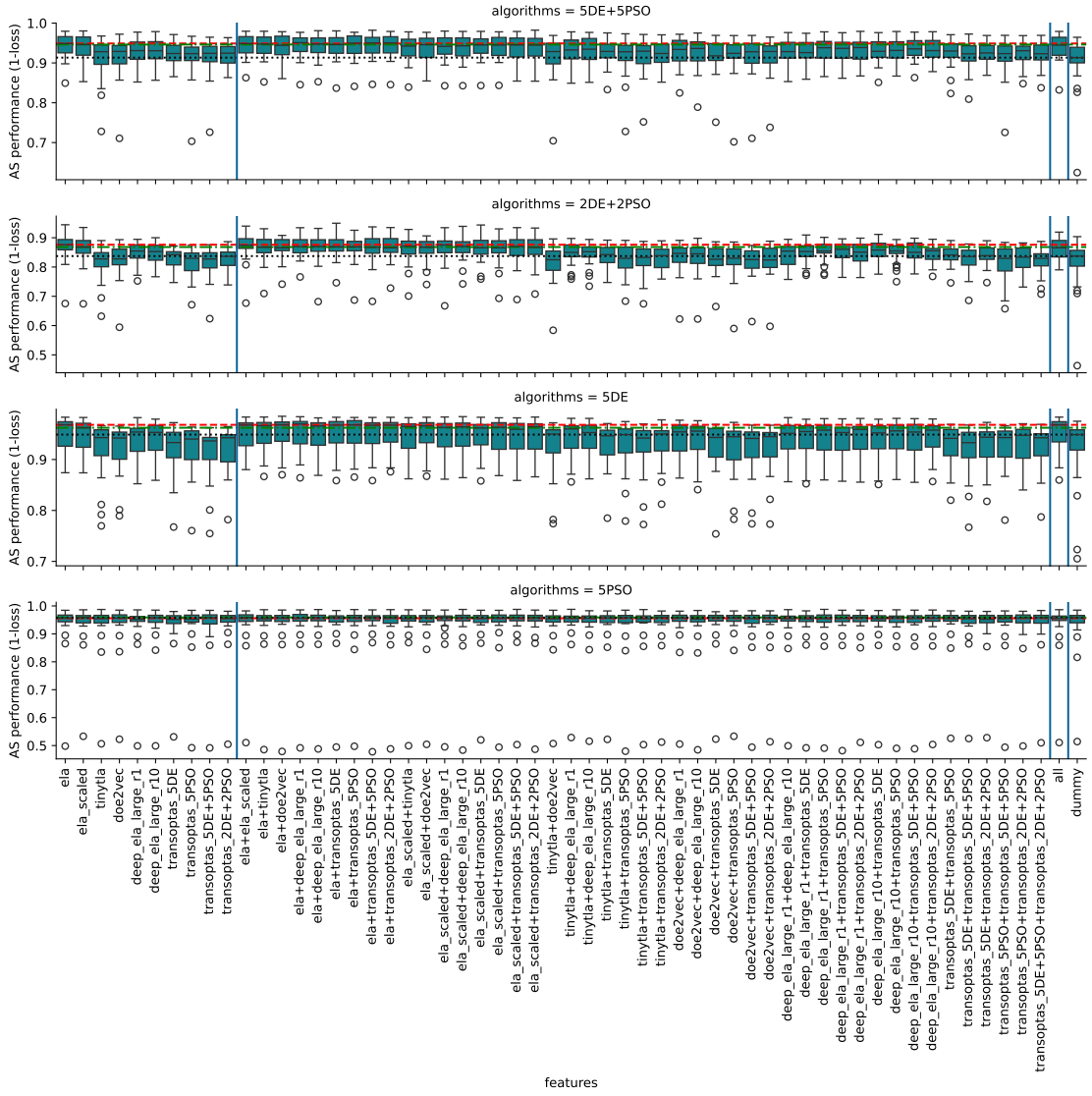



Figure 4.19: AS performance obtained for the different feature groups and algorithm portfolios in the problem combination split evaluation.

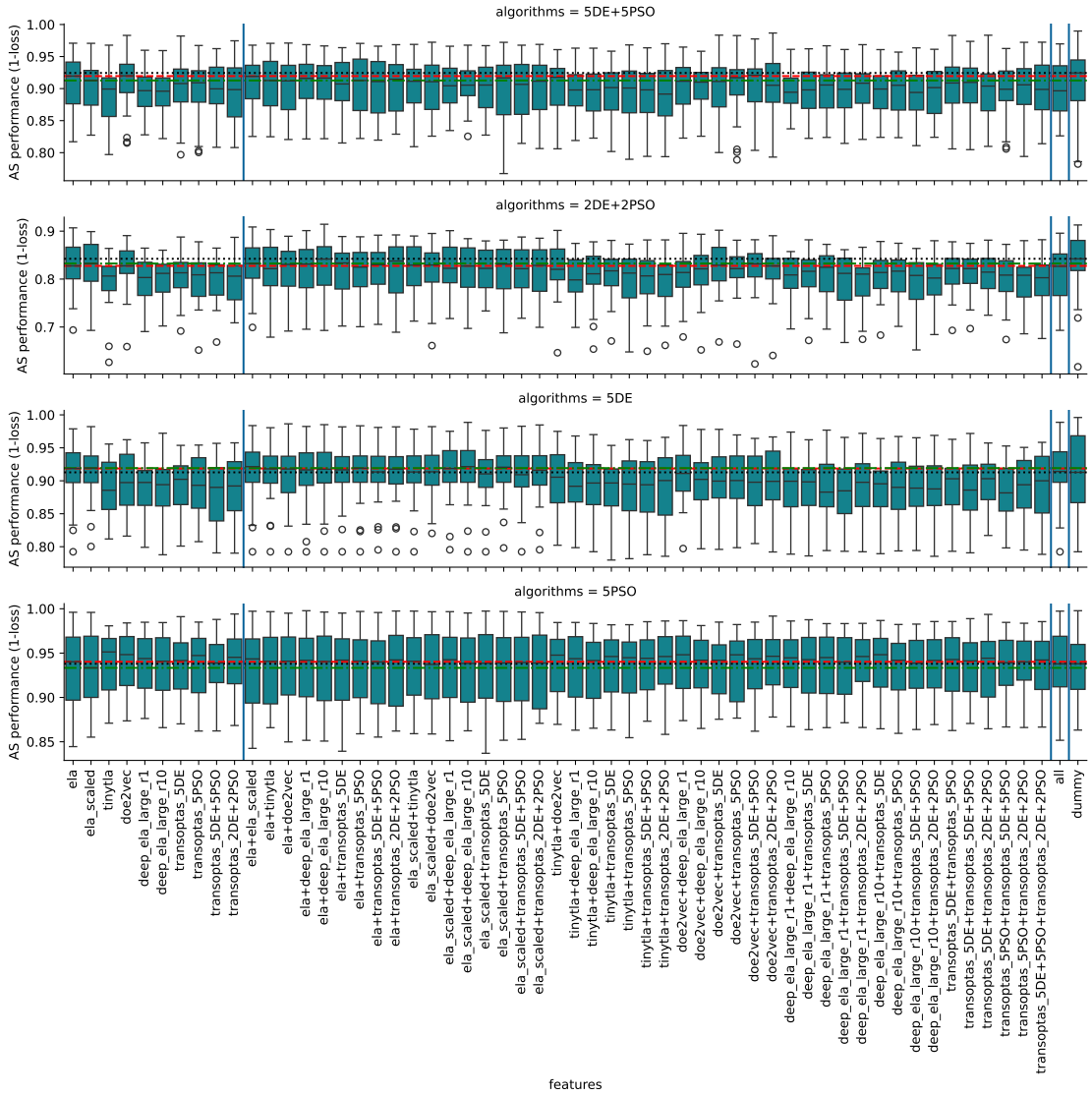



Figure 4.20: AS performance obtained for the different feature groups and algorithm portfolios in the problem split evaluation.

Finally, Figure 4.20 presents the results from the most challenging problem split evaluation scenario, where all problems derived from a pair of problem classes are used for testing. In this case, we can see that the AS is no longer working, with all models having essentially equivalent or inferior performance to the dummy model. The difference between the median AS performance of the *ela* model and the dummy model correnspponds to -0.011, -0.009, 0.006 and -0.004 for the 5DE+5PSO, 2DE+2PSO, 5DE and 5PSO algorithm portfolios, respectively.

### 4.2.3 Discussion

The results of this work show that while new representations for optimization problems have been developed, moving beyond the traditional ELA features, the challenge of generalizing to new, unseen problems persists. Meta-models work well on functions that are the same or very similar to the training data, but they still fall short of the main goal - generalizing

to entirely new functions.

Considering the fact that *doe2vec*, *deepela* and *transoptas* features are trained using different training strategies, and the *tinytla* features use a completely different approach to feature calculation, the obtained results may be an indicator that the features themselves are not the cause of the observed lack of generalizability to unseen data. On the other hand, our experiments show that the similarities between problem instances is quite high, regardless of the feature representation used.

This raises the question of what exactly is preventing the AS model from transferring the knowledge to new data. Our analysis of performance alignment to problem landscape features indicated that alignment between similarity in features and similarity in performance is only clearly observed when feature similarity is extremely high. This could explain the poor generalizability of the meta-model, since the AS model can reliably infer performance if it has previously observed a similar or almost identical problem instance. Other potential causes to be further investigated include:

*i)* Is a $50d$ Latin Hypercube sample of the problem sufficient to capture algorithm performance? Topics to be investigated here from an experimental setup point of view, include both the size of the sample and the sampling technique. Although the sensitivity of ELA features to the sample size and sampling method has been investigated in several works [6], [7], [81], to the best of our knowledge, this topic has not been investigated from an AS generalizability point of view. Since most of the AS works use the instance split evaluation, the sample size of $50d$ has been accepted as the general recommendation. It could be the case that this sample size is sufficient for the easier instance split evaluation, but not for the more challenging evaluation. However, a larger sample size of $1000d$ for calculating the ela features has been used in [9], and in this case, the AS model still did not work in the more challenging problem split evaluation. In light of these observations, trajectory-based features, extracted from the actual behavior of algorithms during the optimization process, may provide complementary information that is not captured by static, sample-based features.

*ii)* All of the investigated feature groups based on deep learning models (*doe2vec*, *deepela* and *transoptas*) leverage a random function generator for creating the training data. The functions generated in this way have been shown to cover a much different area of the problem space compared to the affine functions [120], [142]. This discrepancy in the training data may be an additional factor influencing the AS generalizability. The improvement of deep learning models for feature extraction necessitates a further effort in developing large and diverse training benchmarks which provide a broad coverage of the problem space.

*iii)* One possible explanation for the lack of generalizability is the design of the BBOB benchmark and its current use in the algorithm selection community. While BBOB problems are intended to be diverse, instances of a specific problem are often quite similar to each other. With that in mind, it is possible that no benchmark currently exists that would allow for a fair comparison of features, as one LIO approach might be trivially solved, while LPO is extremely challenging. This could potentially mean that, with existing benchmarks, it is not possible to measure generalizability properly.

The previously provided points are directions towards understanding why this experimental setting is failing. However, the overall results indicate that a more radical change is required in the way AS is approached for continuous SOO, especially in the way it is evaluated.

Finally, we would like to acknowledge a limitation of our experimental evaluation, which is the choice of benchmark suite. Namely, we use the affine problems generated by combining problems from the BBOB [18] benchmark. While we consider the affine

problems to provide an improvement over the BBOB benchmark in terms of the quantity and diversity of the generated problems, we would like to point out that they are likely not sufficient to accurately model the vast majority of real-world problems. In fact, several studies indicate that the generated affine problems are not too different from the base BBOB problems used for their construction. This has been observed by investigating the ELA representations of BBOB and affine problems, where it is shown that problems from both benchmarks have feature values in similar ranges [120], [148] and they cover the same area of the problem landscape when visualized in two dimensions [120], [148].

Unfortunately, to the best our knowledge, a sufficiently large benchmark of real-world problems is currently unavailable. Having all of this in mind, we still opted for using the affine combinations as they offer the creation of additional problems in a controlled setup, which are suitable for use in machine learning settings and evaluating the generalization of AS models. We do acknowledge the impact the training data has on our evaluation and conclusions, and would like to stress the importance of evaluating AS models on real-world problems, once they become available.

## 4.3 Conclusion

This chapter highlights the limitations of current feature-based AS models in generalizing to out-of-distribution problems within continuous SOO. Despite the advancements in landscape analysis and deep learning-based features, when faced with unseen problems, these models fail to surpass the performance of a simple baseline model. This underscores the need for further research in the rethinking and development of more robust AS models capable of effectively handling novel problem instances.

In this chapter, we investigated hypotheses H2 and H3. Considering the overall results, we can find supporting evidence for H2, saying that the generalization of an AS model across benchmark suites depends on the extent of problem coverage provided by the benchmark and the distribution of algorithm performance. On the other hand, our results lead to the rejection of H3, which posits that recently developed landscape features and their combinations outperform traditional landscape features for the AS task. This is evident by the fact that none of the investigated features (*transoptas*, *doe2vec*, *deepela* and *tinytla*) outperform the ELA features or contribute to a performance increase when combined with the ELA features.

# Chapter 5

# Trajectory Features

Previous chapters have focused on the proposal and evaluation of problem landscape features, which are typically calculated using a set of solution candidates which are sampled from the decision space of the optimization problem instance via some non-adaptive approach such as uniform sampling or Latin Hypercube sampling and which are then evaluated. Therefore, these methodologies generate a "static" depiction of the problem that is not tailored to a specific algorithm. The representations embody characteristics of the entire problem space, regardless of the sections of the problem space that have been encountered by the algorithm in question during its search procedure. It has been demonstrated that the characteristics of the problem space that an algorithm encounters during optimization can substantially differ from the global landscape of the problem being solved [25]. Additionally, features that capture the *algorithm-problem interaction* can also benefit tasks such as dynamic AS, where the ML model is deciding whether it is beneficial to switch to a different algorithm instance online, during the run of an algorithm instance [107]. Another potential application of features that capture the algorithm-problem interaction is the selection of algorithms for an algorithm portfolio, i.e., the set of algorithms the algorithm selector is asked to choose from. Finally, representing algorithm trajectories in terms of numerical features would also allow the analysis of novelty in newly proposed optimization algorithms by comparing their search behavior to existing algorithms.

Driven by the desire to capture the algorithm-problem interaction, we propose the DynamoRep features for problem representation using trajectory-based population dynamics. The primary goal of this approach is to utilize the simplest and least computationally demanding features that can be derived from the samples of the problem which the algorithm is already examining during its search process. The rationale behind using these features is that they do not add extra computational overhead to the algorithm's execution and do not necessitate additional function evaluations to generate the samples, as the solutions are already being evaluated as part of the algorithm's search. The DynamoRep features are proposed for continuous SOO problem instances, although they have potential to be extended to other domains and settings, such as, for example, a multi-objective one.

**Our contribution:** In this chapter, we provide a comprehensive analysis of the performance of the DynamoRep features for the tasks of **algorithm selection(AS), problem classification, and algorithm classification.** AS is the most common task used to evaluate the predictive power of problem representations, due to its highest practical utility [1], [12], [98], [124]. The problem classification task is well explored in several studies [14], [149] in which different groups of problem landscape features are used to test their utility to describe an optimization problem instance. This task has been performed to find good problem landscape features that in the future can help in the selection of diverse and unbiased problem instances that are involved in the experiments. However, the motivation

for evaluating the DynamoRep features in the problem classification tasks covers other perspectives. First, we can detect problems on which the algorithm instance has similar search capabilities (if it is difficult for the classifier to distinguish between different problems in the classification task) or different search capabilities (if it is easy to distinguish between different problems in the classification task).

In addition, we explore the potential of the DynamoRep features for algorithm classification. The motivation behind this choice of a learning task is that it allows us to evaluate whether the features have enough predictive power to distinguish between the behavior of different algorithms.

They could be further used for *(i)* dynamic AS, *(ii)* selecting algorithms with complementary behavior to be included in an algorithm portfolio used by an algorithm selector, or *(iii)* for analyzing the novelty of different algorithms by emphasizing the differences on which problems the algorithms have different search capabilities and on which problems they have similar search capabilities.

For all three tasks, we investigate the tradeoff between the model performance and the budget used to calculate the features, and conduct a feature importance analysis. We investigate the models' dependency on the range of the objective functions by observing the impact of scaling the objective function values. Additionally, we evaluate the impact of performing differencing of the DynamoRep features with the goal of eliminating the representation's sensitivity to the initial population.

We compare our approach to the most commonly used ELA [5] features in two ways. Firstly, we use the classical way of computing the ELA features on a static sample of problem solutions generated using Latin Hypercube sampling [24], which is independent from the algorithm execution. Additionally, we compare our method to the ELA features computed on the samples from the trajectories generated by the algorithm execution. As opposed to our original work introducing the DynamoRep features [57] and other applications of the ELA features on algorithm trajectory data [150], where the ELA features were calculated by merging the individuals from all of the algorithm iterations into a single sample, here we calculate the ELA features on the samples from each iteration separately, and use the concatenated features from all iterations to represent the trajectory. This has been previously done also in [58], and it is an approach to ensure fair ML benchmarking taking into account the purpose of our comparative study, although it is not the main contribution of this work.

Related works aiming to capture features using dynamic sampling techniques, include features based on the Simultaneous Optimistic Optimization Tree [151] and Search Trajectory Networks (STNs) [152]. In contrast to Simultaneous Optimistic Optimization Tree features [151], which proposes the extraction of features based on a predefined algorithm executed independently of the optimization algorithm used to solve the problem, our approach aims to use the solutions explored by the same algorithm, so that no additional function evaluations are used for feature calculation. Additionally, differently from STNs, whose primary purpose is visualization of algorithm behavior and require multiple runs of the algorithm, the primary goal of our work is to propose numerical representations of trajectories applicable to a single algorithm run, at low cost.

The remainder of the chapter is organized as follows. Subsection 5.1 describes the generation of the DynamoRep features, Subsection 5.2 explains the experimental design, while Subsection 5.3 provides a comparison of the running times to compute DynamoRep and iteration-based ELA features. Subsections 5.4, 5.5, and 5.6 present the results for the AS, problem, and algorithm classification tasks, respectively, while Subsection 5.7 concludes the chapter.

**Reproducibility:** The code for the experiments is available at `https://github.com/`

`gjorgjinac/DynamoRep_ext`. The algorithm execution data and the results of the experiments are available at `https://portal.ijs.si/nextcloud/s/ECaBkzj2QTx4Jkq`.

## 5.1 DynamoRep Representations

Given an optimization algorithm and an optimization problem instance, we aim to generate a vectorized representation of the interaction of the algorithm with the problem instance, which takes into account the algorithm trajectory's longitudinality. The main objective of the approach is to use the simplest, most computationally inexpensive features which can be extracted from the samples which the algorithm is already considering in its search procedure. The reasoning behind such features is that they do not introduce an additional overhead in the algorithm execution, and do not require an additional budget of function evaluations for generating the sample upon which they are calculated, since the solutions are anyway evaluated for the purposes of the algorithm's search.

For the following description, we assume that we are dealing with a $d$-dimensional search space $S \subseteq \mathbb{R}^d$, a SOO problem instance $f : S \to \mathbb{R}$, and a population-based algorithm being executed on the problem $f$ with population size $\lambda$. Denoting by $(x_0^i, x_1^i, ..., x_{d-1}^i, y^i)_{i=1,...\lambda}$ the $\lambda$ decision variables $x^i = (x_0^i, x_1^i, ..., x_{d-1}^i)$ of the population and their function values $y^i = f(x^i)$ obtained in a single iteration, we use as a representation a concatenation of the following expressions
$\big(\min\{x_0^1, \ldots, x_0^\lambda\}, \ldots, \min\{x_{d-1}^1, \ldots, x_{d-1}^\lambda\}, \min\{y^1, \ldots, y^\lambda\}\big)$,
$\big(\max\{x_0^1, \ldots, x_0^\lambda\}, \ldots, \max\{x_{d-1}^1, \ldots, x_{d-1}^\lambda\}, \max\{y^1, \ldots, y^\lambda\}\big)$,
$\big(\text{mean}\{x_0^1, \ldots, x_0^\lambda\}, \ldots, \text{mean}\{x_{d-1}^1, \ldots, x_{d-1}^\lambda\}, \text{mean}\{y^1, \ldots, y^\lambda\}\big)$,
$\big(\text{std}\{x_0^1, \ldots, x_0^\lambda\}, \ldots, \text{std}\{x_{d-1}^1, \ldots, x_{d-1}^\lambda\}, \text{std}\{y^1, \ldots, y^\lambda\}\big)$.
The representation concatenates the coordinate-wise minimum, maximum, mean, and standard deviation, respectively. In this way, we create a representation for a single iteration of the optimization algorithm. A representation for the entire trajectory is then generated by concatenating the representations of all of the iterations. Since we use four descriptive statistics, the representation for each iteration is of size $4(d + 1)$. If $n$ is the number of iterations for which the algorithm is run, the representation of the algorithm trajectory is hence of dimension $4n(d + 1)$.

We have selected these specific descriptive statistics for the aggregation of the dimensions of each population due to their low computational overhead, and the ability to obtain them without additional function evaluations, simply by utilizing the candidate solutions evaluated by the algorithm during its run. With these statistics, we aim to capture information about the nature of the candidate solutions generated by the algorithm (e.g., their range and diversity) in each iteration, as well as their quality (objective function values). Additionally, these statistics are computable for any sample size and can therefore be used for different population size, which is not the case for some ELA features for example, which produce invalid NaN values when the sample size is not sufficient. Furthermore, these statistics are widely utilized as aggregation operators in other ML domains such as computer vision [153] and graph neural networks [154].

## 5.2 Experimental Design

In this section, we outline the crucial components of the experimental design, including the *problem portfolio* (the set of problem instances that the algorithms are run on), the *algorithm portfolio* (the set of algorithms that are run on the problem portfolio), and the *machine learning model* (trained for AS, as well as problem and algorithm classification

based on the proposed features). We also provide a detailed description of the models' evaluation.

### 5.2.1 Problem portfolio

For our empirical evaluation, we use the first 100 instances of each problem class from the BBOB benchmark, in dimensions $d \in \{3, 5, 10, 20\}$, respectively. However, due to space constraints, we mainly focus on dimension five.

### 5.2.2 Algorithm portfolio

The *pymoo* [139] python library is used to run the DE, PSO and ES algorithms on the problem instances from the BBOB benchmark. We run the algorithms in a fixed-budget scenario with a population size of $10d$, and a budget of 50 iterations (including the initial population). A budget of 50 iterations therefore corresponds to $500d$ function evaluations. We use a budget of up to 30 iterations to calculate the features, and use the maximal budget of 50 iterations for AS. In future work, algorithms with smaller population sizes (less information in a single population regardless of the number of features) will be investigated. We need to point out here that the selection of the algorithms is only to test the proposed methodology. In the future, different algorithms can be used depending on the task that is being solved.

All algorithms use Latin Hypercube sampling [24] to construct the initial population. The rest of the configuration properties of the algorithms are set to the default values provided in the *pymoo* library, version 0.6.0. We perform five executions of each algorithm on each of the 2,400 problem instances. To avoid a potential bias with respect to the initialization, the five executions are performed with five different random seeds, which are fixed before the algorithm execution. This means that the algorithms executed with the same random seed have the same initial population.

### 5.2.3 DynamoRep feature calculation

The feature calculation step involves the generation of the DynamoRep features as indicated in Section 5.1. With the goal of investigating the sensitivity of the machine learning model to the range of the objective function values and to the initial population considered by the algorithm, we perform experiments with two optional steps in the feature calculations:

- **Scaling:** In order to investigate the sensitivity of the machine learning model to the range of the objective function values, we perform experiments when the objective function values ($y$ values) are scaled in the range [0,1] as explored in [82]. This is done using min-max scaling on a trajectory level, meaning that the values are scaled within each run of the algorithm on each problem instance, before calculating the DynamoRep features. We keep the decision variables ($x$ values) unscaled in both cases, since in the COCO environment, they are already limited to be in the range [-5,5]. Note that if one would like to check the influence of using a different number of iterations for calculating the DynamoRep features, the scaling should be done separately for each feature calculation budget, since the features should not be scaled on the basis of the range of the objective function values in the entire algorithm trajectory, but only within the range of the observed iterations which are used to calculate the features.
- **Differencing:** Differencing is a technique used to transform time series data by calculating the differences between successive observations [155]. This method can help

stabilize the mean of the series by removing changes in the level, thereby reducing or eliminating trends and seasonal patterns. In our case, the goal of performing differencing is to remove the DynamoRep features' sensitivity to the initial population which is dependent on the random seed used to execute the algorithm. The differencing of the DynamoRep features is performed by calculating the difference between each feature calculated with one statistic on one of the problem dimensions (decision variables or objective function value) and its corresponding feature from the previous iteration. This means that if we have features corresponding to the standard deviation of the objective function value in the first and second iteration $std_it_0_y$ and $std_it_1_y$, these would be transformed into one feature $std_it_1_y - std_it_0_y$.

Note that the scaling step happens before the DynamoRep features are calculated, while the differencing step is performed after the feature calculation.

### 5.2.4 Evaluation tasks

We evaluate the utility of the DynamoRep features for the tasks of AS, problem classification, and algorithm classification. Following the most common setup in the analysis of feature-based approaches to black-box optimization [5], [14], [150], [156], we use RF models which we train for the three tasks, given the DynamoRep features extracted from the trajectory of the algorithm run. The models are evaluated using stratified 10-fold cross-validation. The RF model was selected since it offers good performance on tabular data [140] and it provides feature importance values then help with interpretability. The way the data is split into training and testing sets depends on the task the model is trained for. For all tasks, the RF models are executed using the default configuration parameters in the *scikit-learn* [146] python library version 1.2.2. We do not perform parameter tuning in order to evaluate the impact of the proposed features only, on a fixed model configuration.

#### 5.2.4.1 Algorithm selection

We treat AS as a regression task, where the RF model is trained to predict the performance of the three optimization algorithms on the basis of a single execution of one of the algorithms. In this AS setting, we aim to evaluate whether the trajectory features derived from a single algorithm provide enough information to predict its performance after additional iterations, as well as that of other algorithms, which will then be used to identify the most suitable one. This means that one of the algorithms is executed for some feature calculation budget of $b$ iterations and the features are calculated up to this iteration. The performance of the algorithms is then predicted for a total budget of $B > b$ iterations. A single RF model is trained for each problem dimension, each different $b$ (feature calculation budget) and for each of the three algorithms. In our experiments, we only consider one total budget $B$ of 50 iterations (for which we are prediction algorithm performance), with varying feature calculation budgets $b \in \{1, 3, 5, 10, 20, 30\}$.

To capture the algorithm performance, we use the normalized precision score introduced in [120], and later used in [98], [142].

We evaluate the AS model in two settings:

- Instance-split - In this evaluation setting, the data is split into training and testing sets in such a way that instances from the same problem classes are present both in the training and in the test set. In each fold, the runs that belong to 10% of all problem instances from each problem class are removed from the training set and used for testing.

- Problem-split - In this evaluation setting, the data is split so that the training and testing data always consist of instances from different problem classes, i.e., instances from the same problem class cannot appear both in the training and in the testing data. In each fold, the runs of 10% of all problem classes are for testing, and the runs of the remaining 90% of the problem classes for training.

We evaluate the performance of the RF model by considering the true score of the algorithm which was selected by the model for each problem instance:

$$AS_performance = \frac{1}{|\mathcal{P}|} \sum_{p \in P} 1 - (s_{sp} - s_{bp}) \tag{5.1}$$

Here, the true score of the algorithm selected by the RF model for problem instance $p$ is denoted by $s_{sp}$, while $s_{bp}$ is the score of the actual best-performing algorithm for problem instance $p$. This performance metric is a maximization metric bounded between zero and one. A score of zero is obtained if the worst-performing algorithm is consistently selected as the best for all problem instances, whereas a value of one is attained when the algorithm selector correctly identifies the best algorithm for every problem instance. The score is meant to capture how much one is losing by selecting an algorithm which is not the ideal one for that problem instance.

#### 5.2.4.2 Problem classification

In the case of a problem classification task, the RF model gets as input the DynamoRep representation generated from the trajectory of one execution of an optimization algorithm and predicts to which of the 24 problem classes from the BBOB collection the representation belongs to. We train a separate RF model to classify problem instances from each problem dimension, and each of the three optimization algorithms (DE, ES, and PSO). The split of the data into training and testing sets is done on an instance level. In each fold, the runs that belong to 10% of all problem instances from each problem class are removed from the training set and used for testing. We train each model on the trajectories obtained with a single algorithm execution (one random seed) and test it on trajectories obtained with 5 different seeds, to account for the fact that in practical applications, the algorithms might be executed with random seeds that are different from the one the model was trained on. That is, the models use 2,160 training instances (1 algorithm $\times$ 1 algorithm execution $\times$ 24 problems $\times$ 90 instances), and 1,200 testing instances (1 algorithm $\times$ 5 algorithm executions $\times$ 24 problems $\times$ 10 instances). One important thing to note is that for higher feature calculation budgets and high problem dimensions, the number of features can exceed the number of training instances. This can be further reduced using some dimensionality reduction techniques, however, in this case, we do not perform any dimensionality reduction so we can evaluate the importance of each feature separately. However, when used in practice, dimensionality reduction methods should be applied, especially in the case of a limited number of training instances or high problem dimensions.

#### 5.2.4.3 Algorithm classification

In the algorithm classification task, the input of the RF model is the same as for the problem classification task, however, the RF model is trained to predict to which of the three algorithms the trajectory belongs to. In this case, we train a single RF model for each problem dimension. For this task, the problem instances are split on a problem class level, i.e., each fold uses the runs of 10% of all problem classes for testing, and the runs of the remaining 90% of the problem classes for training. This means that in each fold, the

runs of all problem instances from selected 2-3 problem classes (200-300 instances) will be removed from the training data and used for testing.

In the folds where 3 problem classes are used for testing, the models are trained on 6,300 instances (3 algorithms $\times$ 1 run $\times$ 21 problems $\times$ 100 instances) and evaluated on 4,500 instances (3 algorithms $\times$ 5 run $\times$ 3 problems $\times$ 100 instances).

The goal of such a split is to determine if the model can identify an optimization algorithm from its sampling dynamics on previously unseen problems. Consistent with the problem classification task, we train each model with trajectories obtained with one algorithm execution (single random seed) and testing with trajectories obtained with different executions (different random seeds).

### 5.2.5 Comparison to ELA features

We perform the comparison to the ELA features in two ways: *i)* calculating the ELA features on the samples generated in each iteration of the algorithm execution, and *ii)* calculating the ELA features in the classical way, on a "static" sample of solutions generated using Latin Hypercube sampling [24]. Both approaches are explained in detail in continuation.

#### 5.2.5.1 Iteration-based ELA features

For all tasks, we compare the performance of the DynamoRep features to ELA features calculated on the population from each iteration of the optimization algorithms' run. The comparison with the ELA features calculated separately for each iteration in the section was included to ensure a fair experimental setup for feature comparison. While we recognize that ELA features were not originally intended for iterative use, from a machine learning perspective, two feature representations are considered comparable when they are derived from the same data.

We compute ELA features from the following categories using the *flacco* R library [69]: *basic*, *disp*, *ela_distr*, *ela_level*, *ela_meta*, *ic*, *nbc*, and *pca*. In this way, we calculate a total of 93 features. We discard all features that have more than 10% missing values in the training set, while for the remaining features, the missing values are imputed with the mean value of the feature in the training set. We need to point out here that, instead of the mean, the median value can be used as the most robust statistic, however, we have not performed it since the classifiers obtain good performance by using the mean value.

The ELA features calculated for each iteration are concatenated together to form a representation for the whole algorithm trajectory which is given as input to the RF model.

It may be important to note that when the DynamoRep features were originally introduced [57], the comparison to the ELA features was not performed in the same way. Namely, the samples from all iterations of the algorithm's trajectory were concatenated, and the ELA features were calculated on the entire trajectory as was previously done in [150]. Here, instead, we calculate the ELA features on each iteration separately, and we then concatenate the representations for each iteration. This is due to the fact that in the original study, we only used $3d$ problems and a population size of 30, which would mean that if the ELA features were calculated on an iteration level, we would only have a sample size of 30.

Due to computational costs associated with calculating ELA features from each iteration, we perform this comparison only for $5d$ problems.The computational complexity of calculating the ELA features for higher dimensions is further increased because we are also testing the influence of scaling the objective function values. This is computationally expensive since the objective function values are scaled for each feature calculation budget

separately, meaning that the ELA features produced for the same iteration of an algorithm trajectory can differ depending on the feature calculation budget.

#### 5.2.5.2 Static ELA features

For the problem classification and AS tasks, we compare our results with the classical approach of computing ELA features on a collection of samples obtained using Latin Hypercube sampling [24]. We refer to this approach as the "Static ELA" features, since the samples they are calculated on do not have a dynamic componenet from the algorithm behaviour. We use the same feature groups as for the Iteration-based ELA features. The Static ELA features are calculated on samples of different sizes: $10d$, $30d$, $50d$, $100d$, $200d$, and $300d$, where $d$ is the problem dimension. These samples sizes are equivalent in terms of function evaluations to running 1, 3, 5, 10, 20, or 30 iterations of an algorithm, since the population size is $10d$. We additionally investigate the impact of scaling objective function values in the range [0,1] using min-max scaling performed before feature calculation.

Please note that for the algorithm classification task, we do not include a comparison to the classical ELA approach of calculating the ELA features on a sample obtained using Latin Hypercube sampling. This is due to the fact that the classical ELA features do not capture algorithm behavior, i.e., they represent the problem instance, regardless of the algorithm being executed. That is, the features remain the same regardless of the algorithm; it is therefore not possible to train a classification model to predict the algorithm from such features. We then train the RF model using the same setup as the DynamoRep features, except here we do not use any algorithm-specific data, only features extracted from the artificially generated problem samples.

We need to highlight that feature selection techniques have not been applied neither to DynamoRep nor to ELA features. This has been done since we are testing the predictive power of different features and are not interested in finding the best classifier. Since the classifier results with all features are very promising and high, applying feature selection can only increase the accuracy by a few percentage points. Even more, since the RF algorithm is used as a classifier in both learning tasks, it is well known that it works by selecting only the important features from a given feature portfolio. Further, this will allow us to find the most important features together with the information from which part of the trajectory they have been calculated. However, in all experiments, the number of features is lower than the number of instances involved in the training data to avoid overfitting.

## 5.3 Computation Time of the DynamoRep and Iteration-based ELA Features

In this section, we analyze the computation time for the DynamoRep and the iteration-based ELA features. Figure 5.1 compares the average time needed to generate the features for one algorithm trajectory consisting of 30 iterations. The running time of the feature calculation procedure was measured on the trajectories of the DE algorithm executed on one problem instance from each BBOB problem class for each of the problem dimensions 3, 5, 10, and 20. The ELA features were computed for each iteration independently, and the total time for a single trajectory is taken to be the sum of the time needed to compute all iterations.

For the DynamoRep features, we explore two implementations. In the first one, the features are extracted from the entire trajectory at once, while in the second one, the features are extracted from each iteration separately and are then concatenated together.

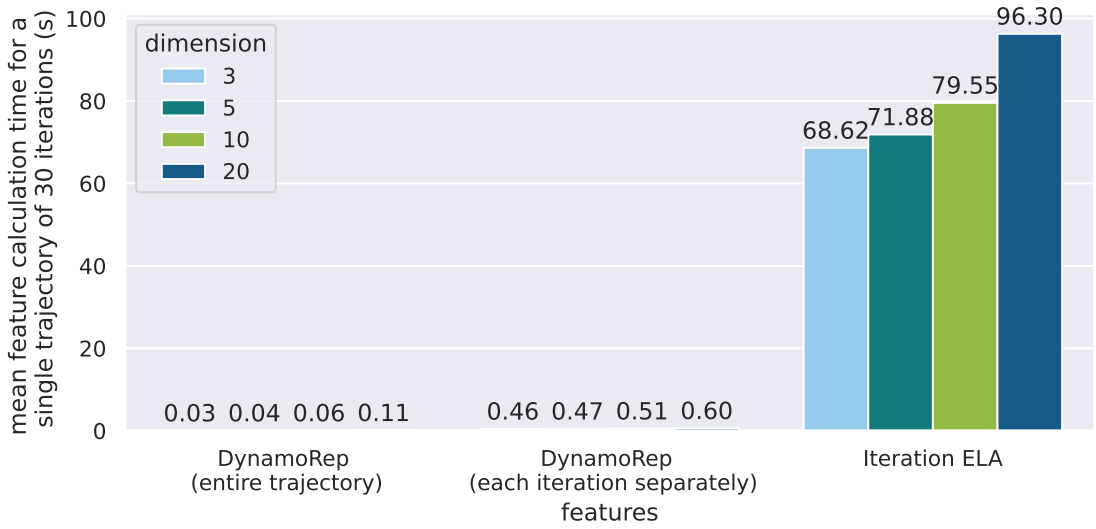



Figure 5.1: Mean time in seconds needed to calculate the DynamoRep features and the iteration-based ELA features for a single algorithm trajectory consisting of 30 iterations.

Both approaches produce the same results, they only differ in the implementation. In our further experiments, we use the first approach, since it is faster due to efficient calculations provided by the *pandas* python library, however, we still include the computation time of the second approach, as it is more similar to the way the iteration-based ELA features are computed and may thus be a fairer comparison. In Figure 5.1, the first approach is denoted as "DynamoRep (entire trajectory)", while the second approach is denoted as "DynamoRep (each iteration separately)". From the figure, we can see that the execution time of both implementations of the DynamoRep features is substantially lower than the iteration-based ELA features. Even for the $20d$ problems, the first approach to calculating the DynamoRep features takes 0.11 seconds, and it is 875 times faster than the 96.30 seconds needed for the iteration-based ELA features. The execution time for a single iteration can be approximated by dividing the values provided for the "DynamoRep (each iteration separately)" and "Iteration ELA" columns by 30, which represents the number of iterations.

One important thing to point out is that the DynamoRep features are implemented in python, while for the ELA features, we use the R implementation of the *flacco* package, so the execution time is also dependent on the programming language. We use the R implementation of the *flacco* package, since the python implementation was not available at the time.

## 5.4 Algorithm Selection Results

In this section, we present the results from the AS task. We start with an analysis of the performance achieved using increasing feature calculation budgets, which we compare to the performance of the iteration-based and of static ELA features. We then analyze the importance of the DynamoRep features for this task.

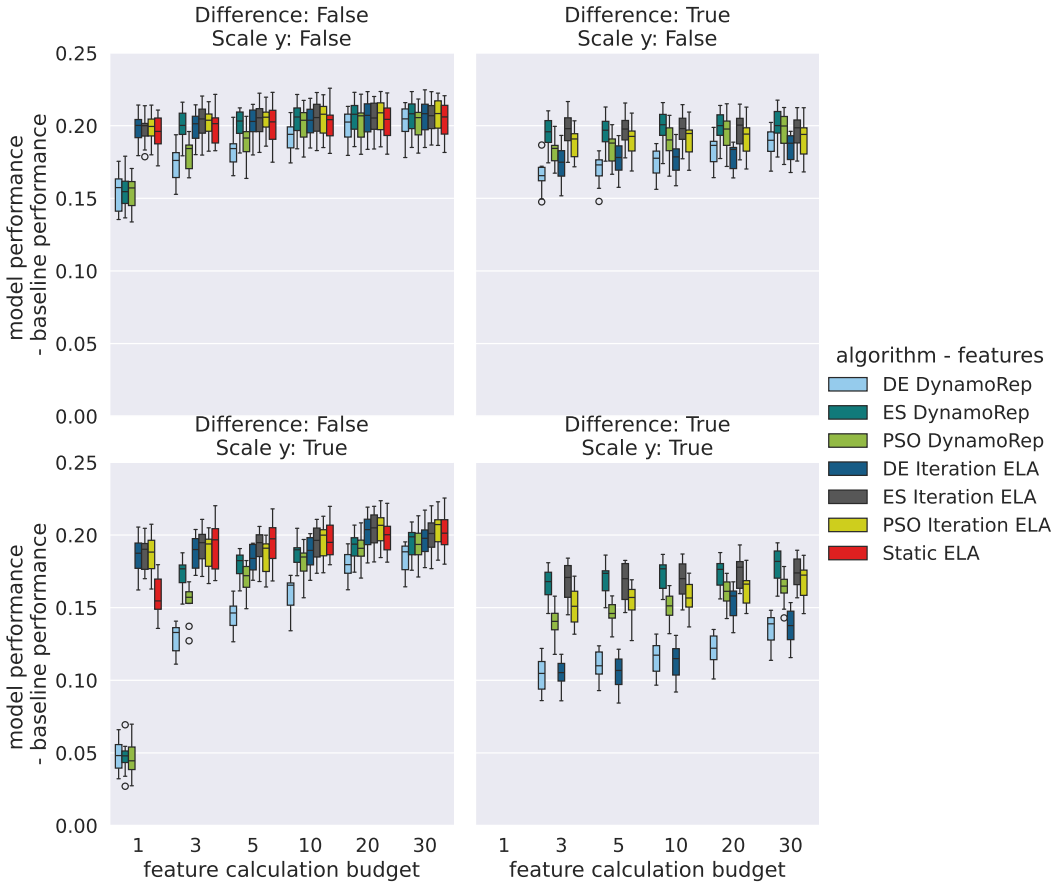



Figure 5.2: AS results using the instance-split evaluation setting for $5d$ problems.

### 5.4.1 Algorithm selection with limited feature calculation budget

Figures 5.2 and 5.3 show the results of the AS task, for the instance-split and problem-split evaluation settings, respectively. The feature calculation budget is denoted on the horizontal axis. The vertical axis shows the difference between the AS performance (as defined in equation 5.1) achieved by the feature-based models and a baseline model predicting the mean performance on the train set for each algorithm (this is equivalent to the single best solver). A score of zero indicates that the feature-based model is equivalent to the baseline, i.e., the features are not useful for AS. The larger the value, the better the model. The differently colored boxplots represent different models. The first three boxplots represent the results of the models using the DynamoRep features extracted from the trajectories of the three different algorithms. The second three boxplots represent the iteration ELA features calculated on the trajectories from the three algorithms. Finally, the last boxplot corresponds to the static ELA features, calculated on the Latin Hypercube sample. Please note here that the function evaluations used for calculating the static ELA features are equivalent to the function evaluations used for the calculation of the DynamoRep and iteration ELA features. Also note that the Static ELA features are only included in the subplots where differencing is not used, since it is not possible to perform differencing on the static sample. Additionally, in the subplots where differencing is used, there are no results for feature calculation budget of one iteration, since a minimum of two iterations is required to perform differencing.

Please note that we are presenting the difference between the feature-based models and

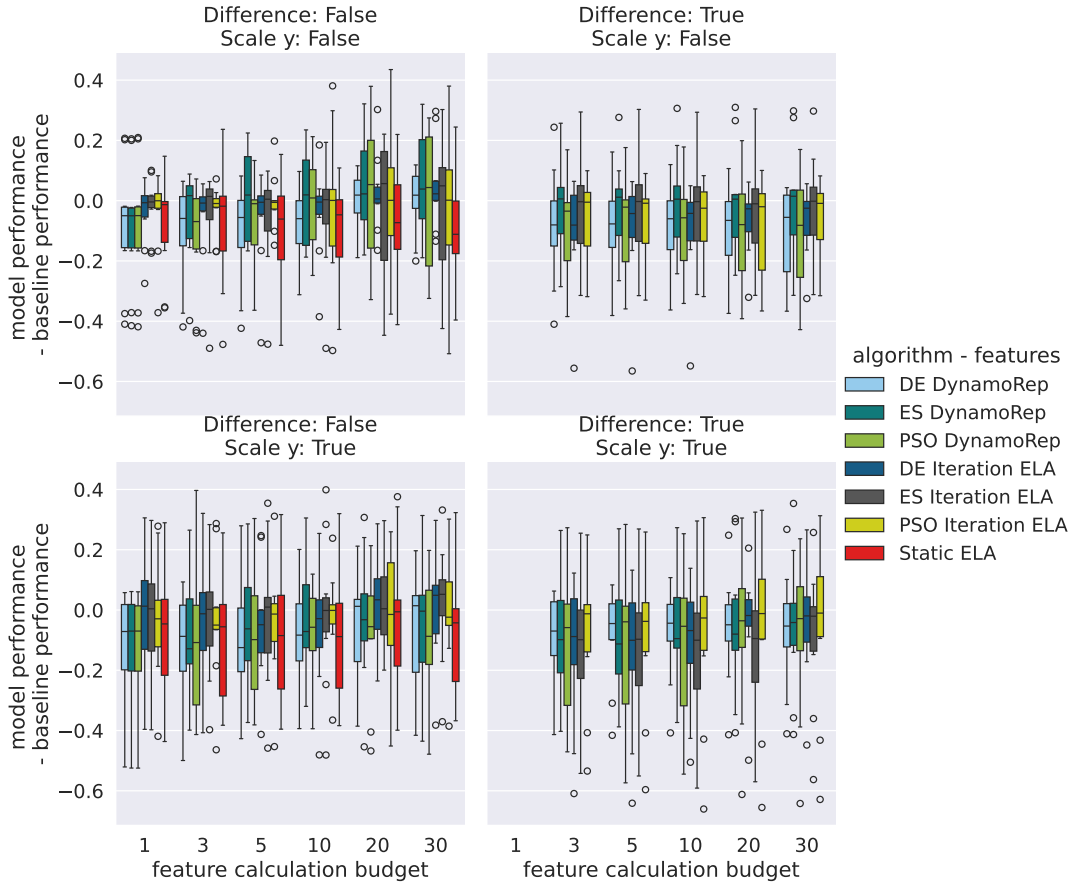



Figure 5.3: AS results using the problem-split evaluation setting for the BBOB problems in dimension 5.

the baseline, since there is a large variation between their performance of different folds, and plotting the actual performance leads to reduced visibility. The variation in performace between folds is especially pronounced for the Problem-split evaluation, where the models are evaluated on previously unseen classes of problems. In this case, the baseline model achieves performances in the range 0.53-0.98 with a median of 0.85 and mean of 0.78. On the other hand, in the Instance-split evaluation, the baseline model has performances in the range 0.75-0.79 with a median of 0.77 and mean of 0.78.

Looking at the results from the instance-split evaluation setting in Figure 5.2, we can see that all of the models outperform the baseline. When differencing and scaling are not performed, the performance of the Static ELA and Iteration ELA features are nearly equivalent. With feature calculation budgets greater than one iteration, the DynamoRep features extracted from the ES algorithm trajectories also obtain comparable results. However, we can see that the results are slightly inferior when using the trajectories from DE and PSO in this setting. This is more pronounced in the setting where scaling is performed, but differencing is not. In this case, we can see that the DynamoRep features extracted from DE trajectories have deteriorated performance, which is not as pronounced for the ES and PSO trajectories. When using a feature calculation budget of a single iteration, however, the DynamoRep features outperform the baseline by only 0.05. This is expected, since the DynamoRep features are meant to capture the algorithm dynamics, not information from a single iteration. The iteration ELA features experience slight performance deterioration when scaling is applied, however, it is not as pronounced as the DynamoRep

features. This can be attributed to the fact that some of the ELA features are not sensitive to scaling [7]. The application of the differencing transformation does not have a large impact when scaling is not applied, however, when the two transformations are used together, they have a negative impact on the performance of all models. Interestingly, the impact of the joint use of differencing and scaling seems to have the least effect on the features calculated on the ES trajectories. Both the Iteration ELA and DynamoRep features calculated on the ES trajectories achieve similar performance, which is superior to the other trajectories.

Conversely, taking into account Figure 5.3, showing the results from the problem-split evaluation, we can see that most of the models do not outperform the baseline. For a few of the models, we can observe that the median is higher than zero, i.e., the model is sometimes outperforming the baseline by a small margin. This is occurring in the scenarios when differencing is not performed, and the feature calculation budget is higher than 20 iterations. However, in these cases, the improvement over the dummy model is negligible and although the median difference exceeds zero, the feature-based models do not consistently beat the baseline, since the performance in some folds is smaller than zero. These results provide a lot of open questions and potential future research directions related to zero-shot AS techniques.

### 5.4.2 Feature importance analysis

Figure 5.4 shows the feature importances assigned by the RF model to the DynamoRep features for different algorithms. The importance of each feature is obtained by taking the median feature importance given by the RF models in each fold in the 10-fold cross-validation. Please note that this analysis is focused on the instance-split evaluation setting, since the models evaluated in the problem-split setting did not outperform the baseline and it is therefore not useful to analyze how they do reasoning. Additionally, we do not analyze the scenarios when the differencing transformation is performed, since it had a negative effect on performance. Each column refers to a different algorithm. The top row refers to the importance assigned by models trained on the original trajectories, without scaling of the objective function values, while the bottom row refers to the importance assigned by models trained on the trajectories with scaled objective function values.

When the objective function values are not scaled, we can observe that the most important features are calculated using the standard deviation statistic. When trajectories from the DE algorithm are used, we can see that most of the selected features come from the later iterations (after iteration 20), or the first iteration. This could explain why the models using the DE trajectories have some predictive power when using a single iteration, however, they reach the performance of the competing models at iterations 20-30. Most of these features are calculated using the standard deviation statistic, which indicates that the RF model learns to do AS based on how the DE algorithm has converged in the final iterations. Considering the ES algorithm, we can see that the most important features are coming from the standard deviations in the initial ten iterations, or the last iteration, which could explain why the DynamoRep models using the ES trajectories are outperforming the DynamoRep models using trajectories from the other algorithms in the initial iterations. For the PSO algorithms, all of the most important features are calculated using the standard deviation statistic, mostly from the first or last ten iterations. On the contrary, when the objective function values are scaled, we can see diverse features having a high importance. For all algorithm trajectories, the top 2 most important features are the standard deviations of the fourth decision variable ($x_4$) from different iterations. The mean of the objective function values from the first and last iterations is also commonly appearing.

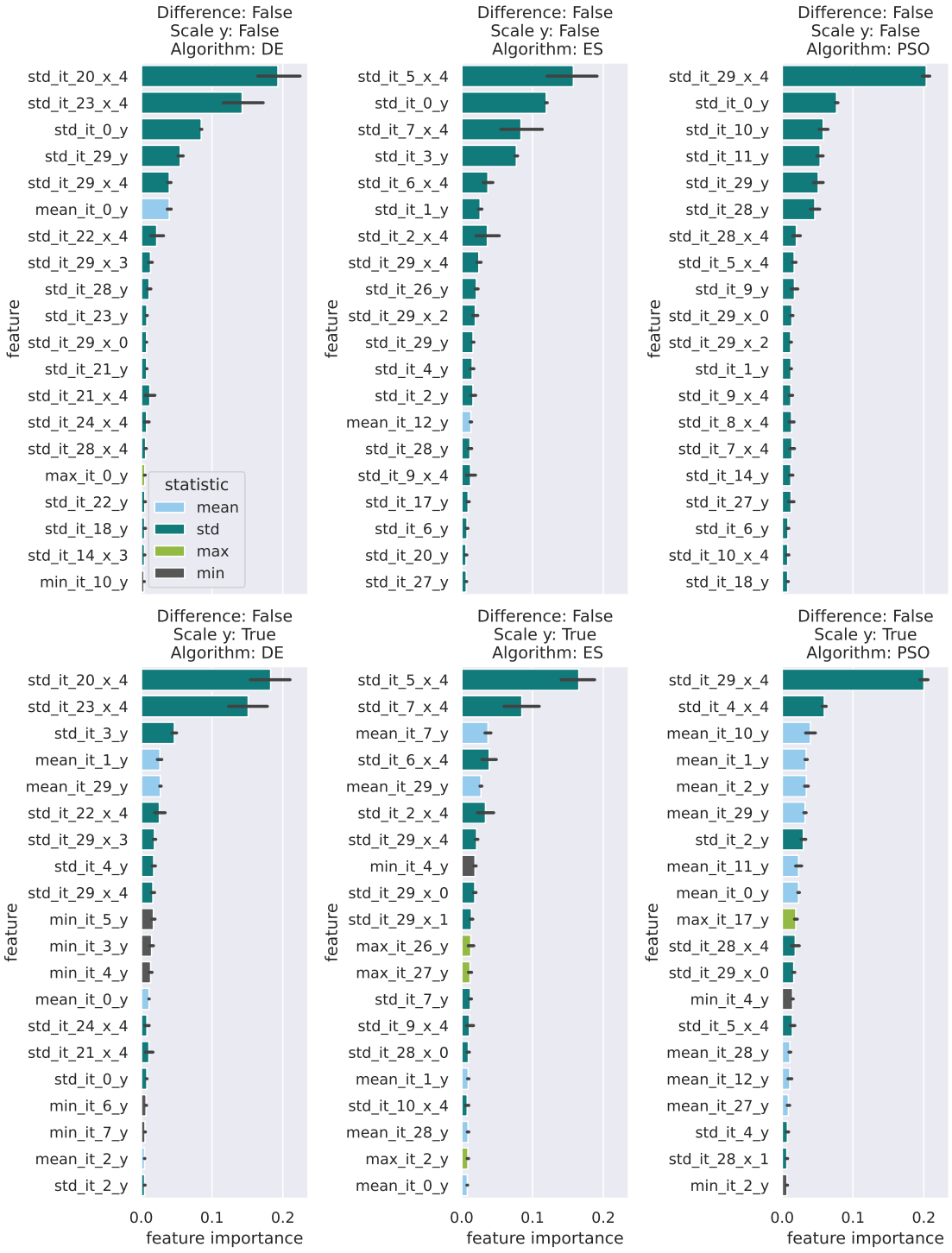
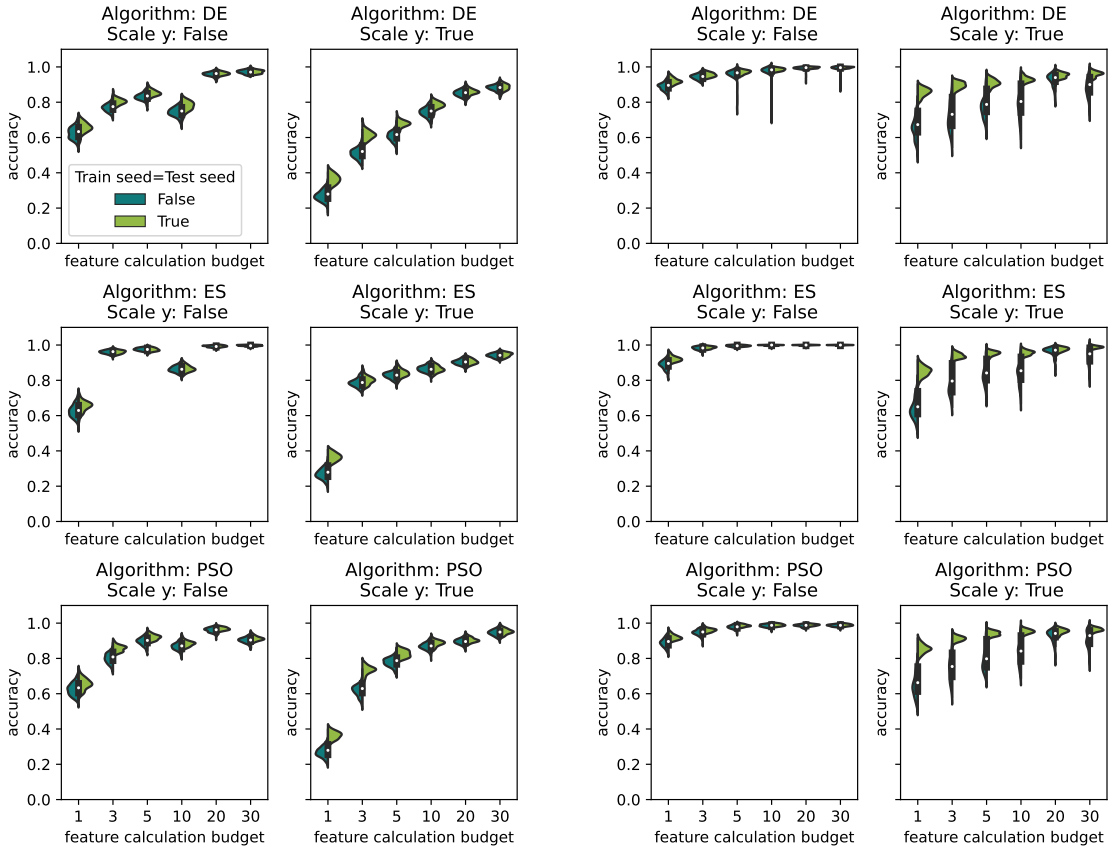



Figure 5.4: Top 20 DynamoRep features with highest feature importance assigned by the RF model for the AS task in the instance-split evaluation setting.




(a) DynamoRep

(b) Iteration ELA

Figure 5.5: Accuracies obtained with the DynamoRep and iteration ELA features for the problem classification task of $5d$ problems.

## 5.5 Problem Classification Results

This section presents the results from the problem classification task, starting with the inspection of the tradeoff between the feature calculation budget and the obtained accuracy, followed by a comparison with the iteration-based and static ELA features, and a feature importance analysis.

### 5.5.1 Problem classification with limited feature calculation budgets

In this subsection, we present the results obtained for the problem classification task, when instead of using the entire algorithm trajectory, the training data is limited to a certain number of iterations. The purpose of this evaluation is to identify at which point of the optimization process, one can train a ML model to identify the problem that the algorithm is solving with sufficient confidence.

Figure 5.5a contains violin plots representing accuracies obtained for classifications of $5d$ problems, respectively.

The application of the differencing step did not result in improved performance for the problem classification task. For this reason, we do not investigate it further.

The subplots in the left column contain the accuracies obtained without scaling the objective function values, while the subplots in the right column contain the accuracies

obtained with the scaling of the objective function values. The y-axis represents the obtained accuracy, while the x-axis represents the feature calculation budget (the number of iterations used to calculate the features that will be further used to train the RF model).

The violin plots represent the distribution of accuracy values obtained across different seeds used for running the optimization algorithm, and different folds in the 10-fold cross-validation of the RF model. The right half of each violin plot (in lime color) refers to the accuracies obtained when the RF model is evaluated on trajectories obtained by running the optimization algorithm with the same random seed which was used for the trajectories in the training data. On the other hand, the left part of the violin plot (in dark green color), represents the accuracies obtained when the RF model is evaluated on trajectories obtained with random seeds other than the one used for training.

The classification accuracy steadily increases by including more iterations for calculating the features, for all dimensions and all algorithms. We can observe that if the objective function values are not scaled, even using a feature calculation budget of one iteration (using the initial population generated using Latin Hypercube sampling, before the algorithm performs any operations), the RF model can already achieve accuracies above 0.5 for $5d$ problems. This means that features obtained from the initial sample of points have relatively high power to differentiate between problem classes. Since different problem classes can have different ranges for the objective function values, they are easily distinguished using simple features. While the BBOB benchmark handles the scaling of decision variables (x-values) because its initial goal was to compare methods that use this information (PSO in particular), the objective function values are not scaled with the same care.

When the objective function values are scaled, the RF model achieves accuracies around 0.2-0.4 when using data from a single iteration. However, increasing the feature calculation budget diminishes the negative effect of the scaling on the model accuracy, with the highest feature calculation budget of 30 iterations providing results that are less than 0.1 lower than the non-scaled model. Using the highest feature calculation budget of 30 iterations, accuracies above 0.8 are obtained in all cases. This result is also a consequence of the benchmark problem instances (BBOB) that are involved in our experiments, which are designed to cover different problem characteristics met in practice, and not to train ML models.

Another observation is that for the ES algorithm, there is a more sudden difference in the accuracies achieved using a single iteration, and using three iterations. Using data from three iterations, the non-scaled RF model already achieves an accuracy above 0.9 for the $5d$ problems.

### 5.5.2 Comparison to iteration-based ELA features

Figure 5.5b features the results obtained for the problem classification task using ELA features calculated from each iteration. Similarly to the previous figures, the y-axis represents the obtained accuracy, while the x-axis represents the feature calculation budget (the number of iterations used to train the RF model). The right half of each violin plot (in lime color) refers to the accuracies obtained when the RF model is evaluated on trajectories obtained using the random seed used for training, while the left part represents accuracies when the model is tested on trajectories obtained with a random seed that was not used for the training data.

From the figure, we can see that when the objective function values are not scaled, the ELA features provide accuracies above 0.80 even with a feature calculation budget of one iteration. However, the scaling of the objective function values disrupts the generalization of the model to different runs of the algorithm. This is evident from the fact that the accuracy indicated by the right parts of the violin plots differs substantially from its left

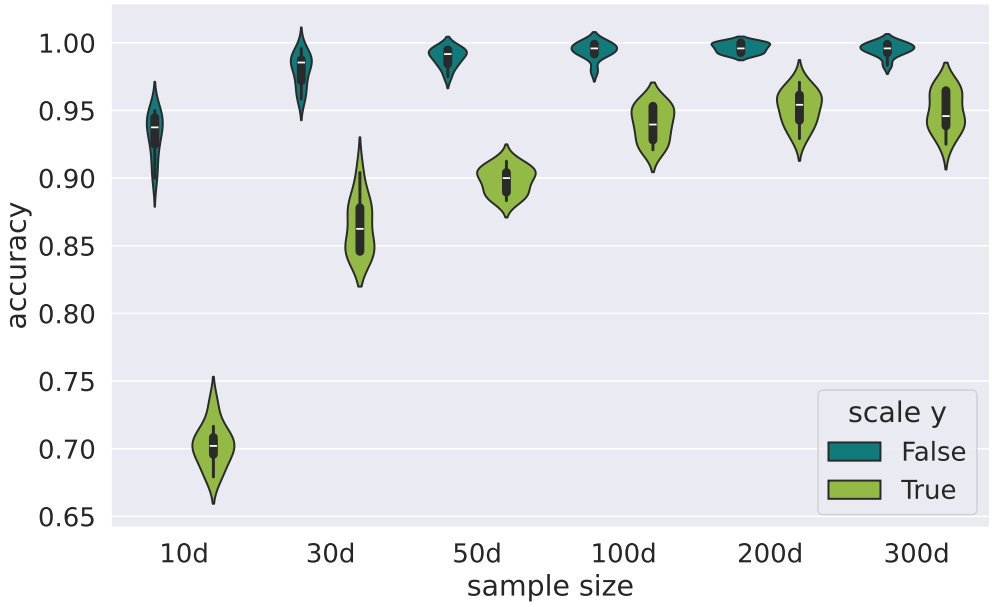



Figure 5.6: Accuracies obtained with the static ELA features for the problem classification task of 5$d$ problems.

counterpart.

In comparison to the DynamoRep features, we can see that the ELA features provide much better results when a lower feature calculation budget is used. However, when higher feature calculation budgets are used, the DynamoRep features provide comparable results. Additionally, when the objective function values are scaled and higher feature calculation budgets are used, the DynamoRep features generalize better to trajectories obtained with different random seeds, while for the ELA features, there is a larger discrepancy between the accuracy on trajectories obtained with the same random seed on which the model was trained, and the accuracy obtained on trajectories with different random seeds.

### 5.5.3 Comparison to static ELA features

Figure 5.6 shows the accuracies obtained for the problem classification task using the static ELA features. The dark green violin plots indicate the results obtained without scaling the objective function values, while the light green ones represent the results obtained when the objective function values are scaled before feature calculation. In this case, we can also observe that the scaling has a detrimental effect on performance, which is especially pronounced when a small sample size is used. As the sample size increases, the features become more robust to this transformation. Compared to the accuracies obtained using the DynamoRep features, we can see that when the DynamoRep features are extracted from trajectories of the ES algorithm, the accuracies are in the similar range, while when the features are extracted from trajectories of the other algorithms, the DynamoRep features provide inferior results. Comparing to the iteration ELA approach, we can see that the results are similar, regardless whether the ELA features were calculated on a Latin Hypercube sample, or on algorithm trajectory samples. However, we have to stress that these methods are not directly comparable, since the data on which the DynamoRep and iteration ELA features are calculated is completely different than the classical ELA ap-

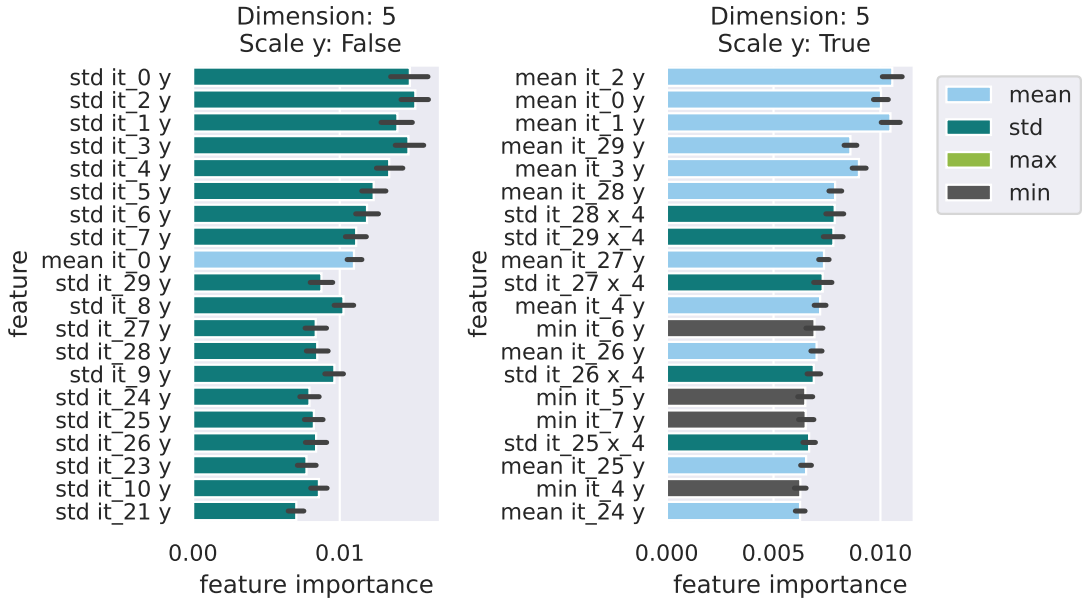



Figure 5.7: Top 20 DynamoRep features with highest feature importance assigned by the RF model for the problem classification task.

proach. The primary goal of the DynamoRep features is to extract useful information using samples obtained during the algorithm execution, while the ELA features requires additional function evaluations which are not used in the algorithm execution.

### 5.5.4 Feature importance analysis

To further understand the value of the DynamoRep features, we analyzed the impurity-based feature importances produced by the RF model trained for the problem classification task. Figure 5.7 shows the 20 most important features. The importance of each feature is obtained by taking the median feature importance produced by the RF models from all folds in the 10-fold cross-validation, across all runs of all three algorithms. In order to include the importance of the features from all iterations, we take the feature importances produced by the models trained with the highest feature calculation budget, i.e., 30 iterations. It is noteworthy to mention that a model trained with a lower feature calculation budget can have different feature importance, since in this case, some of the top 20 features are extracted from the last iterations.

When the values of the objective function are not scaled (left column of the figure), all of the most important features are calculated from the values of the objective function. The standard deviation of the objective function dominates the list of most important features for all dimensions. The mean of the objective function value in the first iteration is among the most important features. However, for higher dimensions, the 20 most important features come exclusively from the standard deviation of the objective function value.

The scaling of the objective function values completely changes the feature importance. As can be seen from the columns on the right, the standard deviation of the objective function does not appear for $5d$ problems, and appears once for $20d$ problems. For $5d$ problems, the standard deviation of the decision variables ($x$ values) from the final five iterations appear as 5 of the 20 most important features. When scaling is applied, the most important features are most commonly the means of the objective function values.

These features appear among the top 20 features 11 times for $3d$ and $5d$ problems, 15 times for $10d$ problems, and 16 times for $20d$ problems. Interestingly, these features appear as the 20 most important features when they are calculated either from the first 10 iterations or the last 10 iterations. This indicates that the most important features are the ones that show how close is the population in terms of fitness values to the individual that is determined to be the best at the end of the optimization.

Figure 5.7 depicts the feature importances aggregated on a dimension level and contains the results for $5d$ problems only. This summarization is provided for a more robust analysis and to be model-agnostic with regard to the optimization algorithm.

## 5.6 Algorithm Classification Results

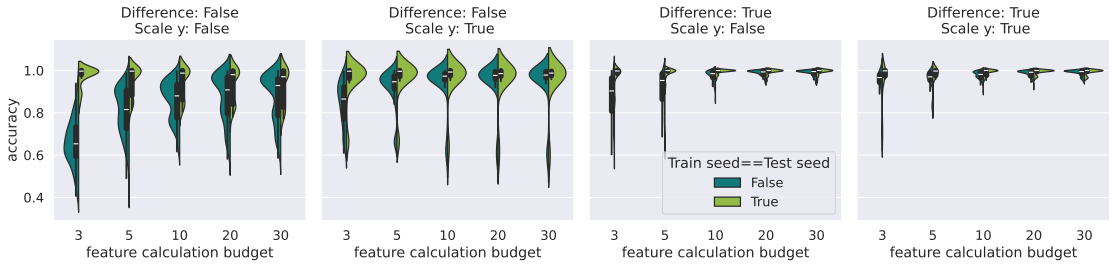
In this section, we present the results from the models trained for the algorithm classification task. The section follows the same organization as Section 5.5.

### 5.6.1 Algorithm classification with limited feature calculation budgets

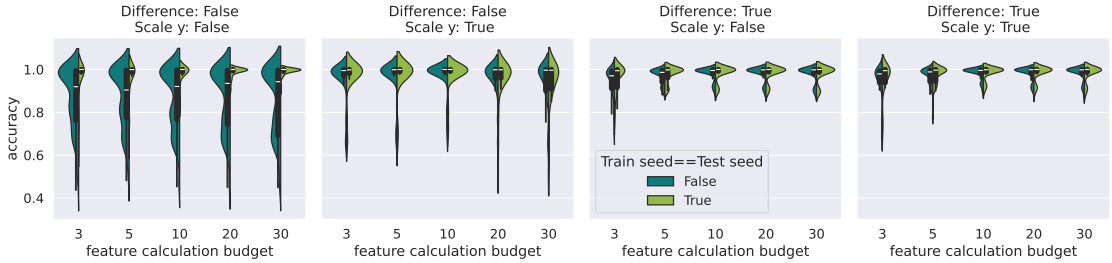
In this subsection, we present the results of the algorithm classification task, exploring different number of iterations used for calculating the DynamoRep features.

Figure 5.8 presents the results from the algorithm classification with different preprocessing steps applied, and using different feature calculation budgets. The rows refer to results for different dimensions, while the columns refer to different combinations of preprocessing steps applied. In particular, the first two columns show the results without differencing the DynamoRep features, while the second two columns contain the results obtained with differencing. The results in the first and third columns are obtained with the unscaled objective function values, while the second and fourth columns are using the scaled objective function values.

The violin plots are split into two parts. The right side of each violin plot (lime color) refers to the accuracies obtained when the testing data contains trajectories obtained with the same random seed as the trajectories used for training. On the other hand, the left side contains the accuracies obtained on the testing set which contains trajectories obtained with different random seeds than the ones used for training. The y-axis depicts the achieved classification accuracy, while the x-axis shows the feature calculation budget, i.e., the number of iterations used to calculate the DynamoRep features. In the case of the differenced features, it is not possible to perform experiments with a feature calculation budget of one iteration, since at least two iterations are required to perform the differencing of the features. For this reason, the results for feature calculation budget one are omitted. In the case of the undifferenced features, the classification accuracy obtained with a budget of one iteration is 0.33, corresponding to random guessing. This is due to the fact that all of the algorithms start their search with the same initial population, i.e., they are represented by the same features, and it is therefore impossible to distinguish between them if only one iteration is used to calculate the features. The scaling and differencing preprocessing steps have a positive impact on the classification accuracy for all dimensions, which is in contrast to the problem classification task, where lower accuracies were obtained with these preprocessing steps. This is due to the fact that the scaling of the objective function values makes the problem classification task more difficult since the model cannot rely on the range of the objective function values to identify the problem. On the other hand, this range is not as relevant for the algorithm classification task, so scaling the objective function values might prevent the model from relying on this information. When the differencing step is applied, the algorithms can be recognized with accuracies above 0.90




(a) DynamoRep




(b) Iteration ELA

Figure 5.8: Accuracies obtained for the algorithm classification task using the DynamoRep and Iteration ELA features, across different experimental settings for $5d$ problems.

for $5d$ problems and nearly perfect accuracy for $20d$ problems with a feature calculation budget higher than 10 iterations. This result also indicates that by applying differencing, the algorithm dynamics is better captured.

### 5.6.2 Comparison to iteration-based ELA features

Figure 5.8b shows the results for the algorithm classification task obtained with the iteration-based ELA features for $5d$ problems. In this case, we can see that the objective function values are not scaled and the features are not differenced (upper left plot), the models do not generalize to trajectories obtained with different random seeds, in some cases achieving accuracy as low as 0.33, corresponding to random guessing. The scaling of the objective function slightly improves the generalization ability. However, the differencing provides a much more substantial improvement to the models' accuracy. When the features are differenced and a feature calculation budget lower than 10 iterations is used, the ELA features achieve comparable performance to the DynamoRep features, while using a higher feature calculation budget, the DynamoRep features provide slightly higher accuracies.

### 5.6.3 Feature importance analysis

Figure 5.9 presents the top 20 features with the highest median importance, for every combination of preprocessing steps. The subplots on the left contain the most important features when the differencing is not applied, while the subplots on the right contain the most important differenced features. The subplots in the first row refer to the features obtained without scaling of the objective function values, while the subplots in the second row refer to the features calculated with scaled objective function values. The feature importance values are aggregated over all dimensions, all folds of the 10-fold cross-validation,

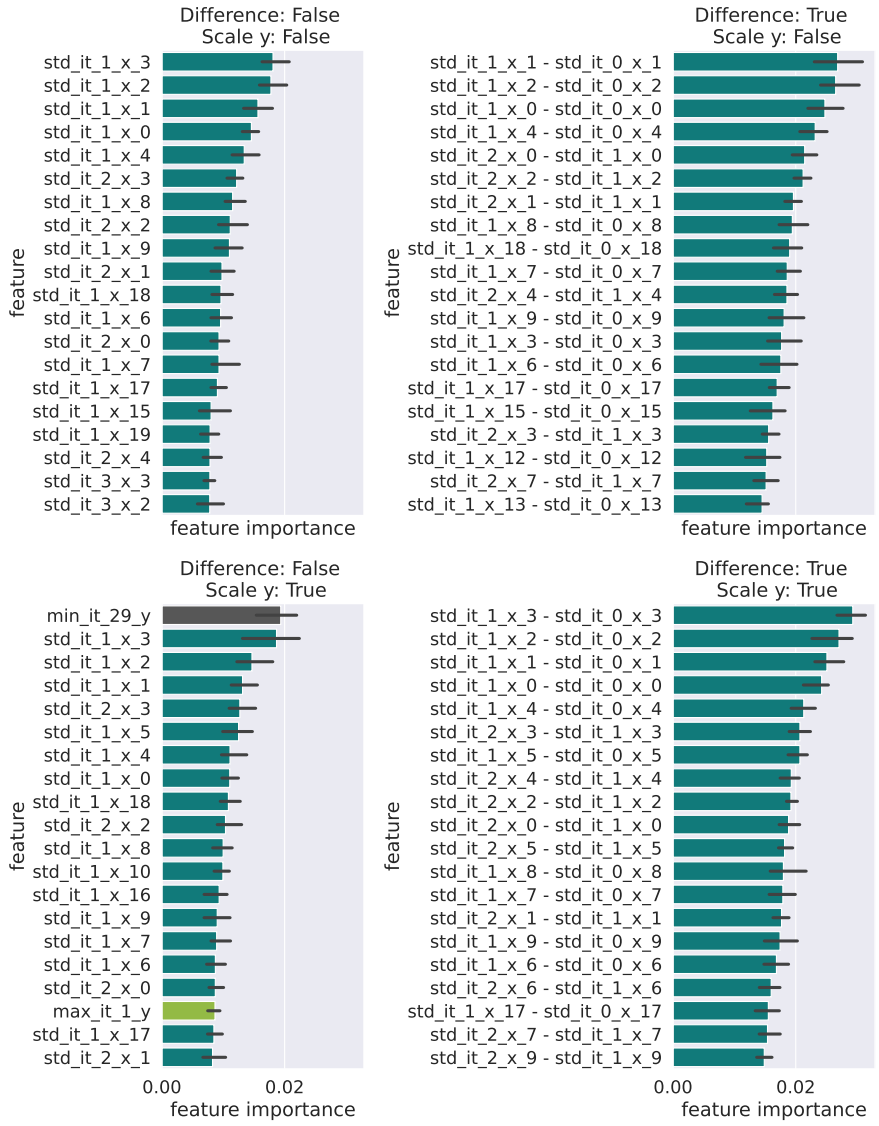



Figure 5.9: Top 20 DynamoRep features with highest median importance for the algorithm classification task, across all four dimensions.

and all models trained on algorithm trajectories executed with a different random seed. From the figure, it is evident that almost all of the most important features are standard deviations of the decision variables, the exception being the minimum and maximum of the objective function value in the 29th and 1st iteration, respectively, which appear among the top 20 most important features when differencing is not applied, and the objective function values are scaled.

## 5.7 Conclusion

In this chapter, we conduct a comprehensive analysis of the DynamoRep features for the tasks of AS, problem classification, and algorithm classification. We evaluate the features' predictive power for all tasks using different feature calculation budgets. The results indicate that the DynamoRep features can be used to distinguish problems and algorithms for problems of different dimensions. The explored preprocessing steps of scaling the objective function values and differencing the DynamoRep features are beneficial for the algorithm classification task, but have a negative impact on the problem classification and AS tasks. The most informative DynamoRep features for the problem classification task are the standard deviations of the objective function values (when the objective function values are not scaled) and the means of the objective function values (when the objective function values are scaled). For the algorithm classification task, the most informative features are generally the standard deviations of the decision variables. A comparison with the ELA features calculated on each iteration of the algorithm execution indicates that ELA features can obtain slightly better performance for the AS and problem classification tasks, at the cost of larger computational complexity, while for the algorithm classification task, the DynamoRep features provide slightly better results with feature calculation budgets higher than 10 iterations.

This Chapter addressed hypothesis H4, positing that simple, low-cost features extracted from the trajectories of optimization algorithms can be effectively used for the AS task, reaching performance on-par or superior to, static ELA features. Our results indicate that the first part of this hypothesis is true, since the DynamoRep features do outperform a baseline model in the instance-split evaluation, however, different outcomes about their superiority over the static ELA features are obtained for different budgets. The DynamoRep features achieve comparable performance to the static ELA features for budgets larger than 10 iterations. However, in the problem-split evaluation, none of the features outperform a baseline model.

As future work, the DynamoRep features can be leveraged for per-run AS and configuration, building on the work of [107]. Their low computational complexity makes them especially suitable for online, dynamic AS. In future work, we intend to explore a scenario where the knowledge obtained from one algorithm is used to warm-start others, enabling AS in a per-run setting by dynamically switching between algorithms. Another direction for future studies is the application of DynamoRep features to larger and more diverse problem and algorithm collections. A particularly interesting possibility is the use of the DynamoRep features for analyzing the novelty of different algorithms by identifying similarities and differences in their search behavior on different problems. They can also be used to select a subset of algorithms with complementary behavior from a given algorithm portfolio used by an algorithm selector. Finally, we are interested in combining problem-focused features such as ELA with DynamoRep features, to exploit their complementarity. We also plan to further investigate the potential of iteration-based ELA, which is a nice by-product of this study, as it can provide useful insights into algorithm behavior.

# Chapter 6

# Discussion and Conclusion

In this chapter, we provide a discussion of the results and conclude the dissertation.

## 6.1 Discussion

This section summarizes and critically reflects on the findings presented throughout the thesis in relation to the four hypotheses (H1–H4) and the three scientific contributions (C1-C3).

### 6.1.1 Discussion of Hypotheses

Each hypothesis addresses a different aspect of the central research objective: advancing the understanding and development of feature representations for Algorithm Selection (AS) in continuous single-objective optimization (SOO). The discussion highlights the degree to which each hypothesis is supported by the empirical results, as well as the broader implications of these findings for the field.

**H1:** *Landscape features learned directly via transformer models - operating on raw sample points from a single-objective continuous optimization problem - can accurately predict algorithm performance without any intermediary feature calculation step*

The results presented in figures 3.8 and 3.9 in Chapter 3 provide evidence in favor of H1. The proposed TransOptAS model outperforms the single best solver in scenarios where there is sufficient complementarity among the algorithms in the portfolio. This demonstrates that transformer-based models are capable of learning meaningful representations of optimization problems directly from raw samples. When compared to traditional Exploratory Landscape Analysis (ELA) features, the transformer-based approach does not yield superior predictive performance. However, the observed differences in predictive power between the two approaches are relatively small (0.02–0.05 in predictive metrics).

Conclusion: H1 is accepted, with the qualification that transformer-based features provide competitive but not universally superior performance compared to ELA.

**H2:** *The ability of an automated algorithm selection strategy to generalize across distinct benchmark suites depends critically on (a) the extent of problem coverage provided by the suite and (b) the underlying performance distribution of the algorithms evaluated on it.*

Chapter 4 examines the performance of AS models trained on different feature sets across distinct benchmark suites. The results confirm the central premise of H2. When benchmark suites share a similar distribution of raw algorithm performance across all algorithms and have the same single-best solver, feature-based AS models exhibit inferior performance-they are outperformed by a simple baseline model that predicts mean performance based on the training data. In contrast, when the training and evaluation bench-

mark suites differ in their single-best solvers, the feature-based models surpass the baseline, highlighting the critical role of algorithm complementarity and performance diversity for effective generalization.

**H3:** *Novel-developed landscape features and their combinations outperform traditional landscape features in automated algorithm selection.*

Conversely, the results presented in Figures 4.17- 4.20 do not support H3. Despite the introduction of several novel feature representations (including deep learning approaches (DeepELA, TransOptAS and DoE2Vec), and topological landscape features (TinyTLA)) none of these feature sets, individually or in combination, outperform traditional ELA features on the AS task. Moreover, combining these new features with ELA does not yield performance improvements. This suggests that while novel features may capture different aspects of the problem landscape, they do not necessarily translate into improved AS performance under the tested experimental setups.

It is important to emphasize the role of the benchmarks used in this evaluation. Due to the absence of more suitable alternatives, an artificial benchmark is employed for the assessment. If a benchmark based on real-world problems becomes available in the future, it would be valuable to re-evaluate the features under those conditions. However, since in the experiments performed, none of the novel features or their combinations surpass traditional ELA features for AS, hypothesis H3 is rejected.

**H4:** *Simple, low-cost features extracted online from the search trajectory of an optimization algorithm can serve as effective inputs for automated algorithm selection, yielding performance on par with—or superior to—existing static feature sets.*

Chapter 5 evaluates the DynamoRep features, which are calculated online during the optimization process. The results support the first part of H4: trajectory-based features are able to outperform a baseline model in the instance-split evaluation, particularly when the feature calculation budget exceeds 10 iterations. However, the second part of H4, which posits superiority over static ELA features, is only partially supported. For the instance-split evaluation of the AS task, ELA features typically perform slightly better, though at a significantly higher computational cost. In problem-split evaluations, neither feature set surpasses the baseline model.

Conclusion: H4 is partially accepted. DynamoRep features offer a promising, low-cost alternative to static landscape features, particularly for online and dynamic AS scenarios, but they do not consistently outperform ELA features across all tasks and settings.

### 6.1.2 Discussion of Scientific Contributions

This section discusses how the results of this thesis address and accomplish the three main scientific contributions (C1–C3). Each contribution is analyzed in terms of the methodological advances introduced, the empirical findings obtained, and their implications for the field of automated AS and landscape analysis in continuous SOO.

**C1:** *Transformer-based feature construction from raw problem samples.*

The first contribution is the introduction of a novel methodology for constructing representations of optimization problems using transformer architectures trained directly on raw sample points from the search space. By leveraging self-attention mechanisms, the proposed models are able to learn expressive features that capture structural characteristics of optimization landscapes without relying on manually engineered features such as ELA. This contribution is realized through the development and evaluation of the TransOptAS model on the AS task, providing evidence for the feasibility of this approach. Even though the TransOptAS model does not outperform the ELA features, the contribution lies in establishing a principled framework for learning problem representations capturing algorithm performance in a data-driven, end-to-end manner, which can serve as a foundation

for future research in learned feature extraction.

**C2:** *Systematic benchmarking study on feature generalizability.*

The second contribution entails a large-scale empirical study that benchmarks the performance of multiple feature families, including traditional ELA features, deep learning-based features (TransOptAS, DoE2Vec, DeepELA), and topological landscape analysis features (TinyTLA). These features are evaluated under a fixed and comparable experimental setup, under increasingly challenging evaluation scenarios. The study establishes new baselines for AS under distribution shift, revealing that existing feature-based models often struggle to generalize to unseen problem distributions. This contribution exposes limitations in current methodologies and highlights the importance of rigorous evaluation when developing AS models.

**C3:** *Introduction of dynamic, trajectory-based features for modeling algorithm–problem interactions.*

The third contribution is acomplished by the introduction of DynamoRep, a set of simple, low-cost dynamic features computed online from the search trajectories of optimization algorithms. It is also complemented by an evaluation of the usage of ELA features calculated on populations of potential solutions considered by the algorithms during its search process. These features capture information about the evolution of algorithm states, providing a dynamic alternative to static, precomputed landscape features.

A comprehensive analysis of the DynamoRep and iteration ELA features is conducted across three tasks: AS, problem classification, and algorithm classification, under different feature calculation budgets. The results show that DynamoRep features outperform a baseline model and achieve comparable performance to ELA features for AS when sufficient iterations are available. These findings highlight the potential of trajectory-based features for online and per-run AS, due to their low computational cost and ability to capture problem-algorithm interactions during the search process.

Taken together, these three contributions advance the state-of-the-art in feature construction and evaluation for automated AS. The transformer-based approach demonstrates that raw landscape information can be effectively encoded without manual feature design. The benchmarking study provides a more rigorous understanding of feature generalizability under distribution shifts, identifying both strengths and weaknesses of existing methods. The introduction of trajectory-based features opens new possibilities for dynamic and online AS, where models can adapt during the optimization process rather than relying solely on static pre-analysis.

## 6.2 Conclusion

This dissertation investigates the role of feature-based representations in AS for continuous SOO, with a particular emphasis on generalization across problem distributions and the practical utility of novel features.

When the work on this dissertation began, ELA features represented the well-established and accepted default choice in feature-based AS for continuous SOO. However, as deep learning rapidly advanced in other scientific domains, several independent research groups recognized the opportunity to develop representation-learning-based features for optimization problems. Within a relatively short time span, a number of different approaches emerged in parallel: autoencoder models trained on reconstructing the sampled objective values (DoE2Vec), transformer models trained in a self-supervised way (DeepELA), transformer models trained for performance prediction (TransOptAS), and topological representations (TinyTLA). Each of these efforts tackled the problem from a distinct perspective and was tested under its own experimental conditions, which ultimately hindered direct

comparison and made it difficult to draw unified conclusions.

The proliferation of these methods marked an important turning point for the direction of this thesis. The central challenge was no longer to simply create new features, but rather to understand, compare, and systematize the performance and limitations of the recently developed ones. Without such comparative analyses, progress risked becoming fragmented, with duplicated efforts and little clarity about what truly advanced the field.

Therefore, the second goal of the thesis became to critically examine deep learning-based and trajectory-based features alongside ELA, asking whether these novel representations genuinely improve generalization across unseen problems. By embedding these comparisons within controlled experimental frameworks and benchmark studies, this work provides the synthesis and evaluation that help to map the landscape of feature-based AS research. In doing so, it moves the field toward a more mature stage, where the focus shifts from parallel invention to cumulative understanding, reproducibility, and eventually, towards the design of hybrid, practical, and generalizable solutions for real-world optimization problems.

Our findings reveal that despite advances in feature representations-including ELA, none of the recently developed features exhibit strong out-of-distribution performance. We posit that this issue likely stems not from the feature representations themselves, but from the fact that AS is an inherently difficult task, and an artificial sample of the optimization problem (which is used as a basis for feature calculation) may not contain enough relevant information to solve it.

Additional aspects influencing our experimental setup and results are the limitations of current problem generation strategies (e.g., reliance on random function generators or affine BBOB combinations), and inadequacies in benchmark design.

An inherent downside of the work done in the AS field in general is the lack of diversity in evaluation benchmarks. Most studies rely heavily on synthetic benchmarks such as the BBOB or on randomly generated problem landscapes, while evaluations on real-world problems remain sparse and fragmented. This lack of diversity has significant implications. First, it limits our understanding of how well feature representations and selection models generalize beyond controlled synthetic scenarios. Second, it creates the risk that models optimized for these artificial settings overfit to narrow distributions of problem characteristics, leading to misleading assessments of their generality. Our analysis of feature-performance alignment further supports this by showing that high alignment is only present in very similar problem instances—highlighting the fragility of current meta-models when extrapolating beyond known landscapes. The field needs benchmarks that are large-scale, diverse, and representative of practical optimization scenarios. Without this structural change, progress in representation learning for AS will continue to be constrained by the artificiality of the evaluation environments.

The insights from this work motivate several critical avenues for future research. First, improving generalization in AS models will require not only new feature representations, but also rethinking benchmark design, evaluation methodologies, and training regimes.

The trajectory-based approach introduced via DynamoRep opens a promising direction for online and per-run AS. In particular, its low overhead makes it suitable for dynamic AS, warm-starting strategies, and even algorithm configuration during runtime. Combining such dynamic features with static ones like ELA could yield robust hybrid models capable of adapting to a broad spectrum of optimization scenarios. The DynamoRep features are only a first step towards the extraction of meaningful information from algorithm trajectories. However, due to their simplistic nature, they capture limited information. This opens up opportunities for the future development of more sophisticated and expressive representations, such as those inspired by deep-learning approaches like Opt2Vec [58].

Another underexplored dimension in the current literature is the interpretability of learned feature representations. ELA features offer some degree of interpretability because they are explicitly engineered to capture known landscape properties (e.g., modality, separability, conditioning). By contrast, deep learning–based representations remain largely black-box, making it difficult to explain their behavior, diagnose failure cases, or design principled hybrid representations. Future work should therefore place greater emphasis on probing, visualizing, and interpreting learned representations, potentially drawing from techniques in explainable artificial intelligence. Interpretability will be essential for building trustworthy and practically useful AS systems, especially in real-world industrial settings where transparency is critical.

Beyond the standard AS approaches presented in this dissertation, recent studies have also explored the use of Large Language Models for automatic algorithm generation, with promising results being reported [157]–[159]. However, our understanding of the generated algorithms is thus far, very limited. As future work, the proposed trajectory features can also be used to analyze the behavior of the produced algorithms and evaluate whether they have a similar search behaviour to existing, man-made ones.

This work contributes both methodologically and empirically to the ongoing effort to develop reliable, efficient, and generalizable algorithm selectors for continuous SOO. It exposes the limitations of current paradigms and proposes novel solutions while emphasizing the broader systemic changes required for sustained progress.

# References


[1] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated Algorithm Selection: Survey and Perspectives," *Evolutionary Computation*, vol. 27, no. 1, pp. 3–45, Mar. 2019, ISSN: 1063-6560. DOI: 10.1162/evco_a_00242. eprint: https://direct.mit.edu/evco/article-pdf/27/1/3/1552398/evco_a_00242.pdf. [Online]. Available: https://doi.org/10.1162/evco%5C_a%5C_00242.

[2] K. M. Malan, "A survey of advances in landscape analysis for optimisation," *Algorithms*, vol. 14, no. 2, 2021, ISSN: 1999-4893. DOI: 10.3390/a14020040. [Online]. Available: https://www.mdpi.com/1999-4893/14/2/40.

[3] F. Zou, D. Chen, H. Liu, S. Cao, X. Ji, and Y. Zhang, "A survey of fitness landscape analysis for optimization," *Neurocomputing*, vol. 503, pp. 129–139, 2022.

[4] K. M. Malan and A. P. Engelbrecht, "A survey of techniques for characterising fitness landscapes and some possible ways forward," *Information Sciences*, vol. 241, pp. 148–163, 2013, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2013.04.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025513003125.

[5] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, "Exploratory landscape analysis," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, 2011, pp. 829–836.

[6] R. D. Lang and A. P. Engelbrecht, "An exploratory landscape analysis-based benchmark suite," *Algorithms*, vol. 14, no. 3, 2021, ISSN: 1999-4893. DOI: 10.3390/a14030078. [Online]. Available: https://www.mdpi.com/1999-4893/14/3/78.

[7] U. Škvorc, T. Eftimov, and P. Korošec, "Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis," *Applied Soft Computing*, vol. 90, p. 106 138, 2020, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2020.106138. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494620300788.

[8] U. Škvorc, T. Eftimov, and P. Korošec, "Transfer learning analysis of multi-class classification for landscape-aware algorithm selection," *Mathematics*, vol. 10, no. 3, 2022, ISSN: 2227-7390. DOI: 10.3390/math10030432. [Online]. Available: https://www.mdpi.com/2227-7390/10/3/432.

[9] G. Petelin and G. Cenikj, "How Far Out of Distribution Can We Go With ELA Features and Still Be Able to Rank Algorithms?" In *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2023, pp. 341–346.

[10] A. Nikolikj, G. Cenikj, G. Ispirova, *et al.*, "Assessing the generalizability of a performance predictive model," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, ser. GECCO '23 Companion, Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 311–314, ISBN: 9798400701207. DOI:

10.1145/3583133.3590617. [Online]. Available: https://doi.org/10.1145/3583133.3590617.

[11] B. van Stein, F. X. Long, M. Frenzel, P. Krause, M. Gitterle, and T. Bäck, "Doe2vec: Deep-learning based features for exploratory landscape analysis," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, ser. GECCO '23 Companion, Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 515–518, ISBN: 9798400701207. DOI: 10.1145/3583133.3590609. [Online]. Available: https://doi.org/10.1145/3583133.3590609.

[12] M. V. Seiler, P. Kerschke, and H. Trautmann, "Deep-ELA: Deep Exploratory Landscape Analysis with Self-Supervised Pretrained Transformers for Single- and Multi-Objective Continuous Optimization Problems," *Evolutionary Computation*, pp. 1–27, Feb. 2025, ISSN: 1063-6560. DOI: 10.1162/evco_a_00367. eprint: https://direct.mit.edu/evco/article-pdf/doi/10.1162/evco_a_00367/2500919/evco_a_00367.pdf. [Online]. Available: https://doi.org/10.1162/evco_a_00367.

[13] G. Petelin, G. Cenikj, and T. Eftimov, "TinyTLA: Topological landscape analysis for optimization problem classification in a limited sample setting," *Swarm and Evolutionary Computation*, vol. 84, p. 101 448, 2024, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2023.101448. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650223002201.

[14] G. Petelin, G. Cenikj, and T. Eftimov, "TLA: Topological Landscape Analysis for Single-Objective Continuous Optimization Problem Instances," in *Proceedings of IEEE Symposium Series On Computational Intelligence*, 2022, pp. 1698–1705. DOI: 10.1109/SSCI51031.2022.10022126.

[15] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010, ISBN: 9781510860964.

[16] N. Patwardhan, S. Marrone, and C. Sansone, "Transformers in the real world: A survey on nlp applications," *Information*, vol. 14, no. 4, 2023, ISSN: 2078-2489. DOI: 10.3390/info14040242. [Online]. Available: https://www.mdpi.com/2078-2489/14/4/242.

[17] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy.

[18] N. Hansen, S. Finck, R. Ros, and A. Auger, "Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions," INRIA, Research Report RR-6829, 2009. [Online]. Available: https://hal.inria.fr/inria-00362633.

[19] A. Nikolikj, C. Doerr, and T. Eftimov, "Rf+clust for leave-one-problem-out performance prediction," in *Applications of Evolutionary Computation: 26th European Conference, EvoApplications 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12–14, 2023, Proceedings*, Brno, Czech Republic: Springer-Verlag, 2023, pp. 285–301, ISBN: 978-3-031-30228-2. DOI: 10.1007/978-3-031-30229-9_19. [Online]. Available: https://doi.org/10.1007/978-3-031-30229-9%5C_19.

[20] J. Kudela and R. Matousek, "New benchmark functions for single-objective optimization based on a zigzag pattern," *IEEE Access*, vol. 10, pp. 8262–8278, 2022. DOI: 10.1109/ACCESS.2022.3144067.

[21] Y. Tian, S. Peng, X. Zhang, T. Rodemann, K. C. Tan, and Y. Jin, "A recommender system for metaheuristic algorithms for continuous optimization based on deep recurrent neural networks," *IEEE Transactions on Artificial Intelligence*, vol. 1, no. 1, pp. 5–18, 2020. DOI: 10.1109/TAI.2020.3022339.

[22] K. Dietrich and O. Mersmann, "Increasing the diversity of benchmark function sets through affine recombination," in *Parallel Problem Solving from Nature – PPSN XVII*, G. Rudolph, A. V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, and T. Tušar, Eds., Cham: Springer International Publishing, 2022, pp. 590–602, ISBN: 978-3-031-14714-2.

[23] D. Vermetten, F. Ye, and C. Doerr, "Using affine combinations of bbob problems for performance assessment," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '23, Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 873–881, ISBN: 9798400701191. DOI: 10.1145/3583131.3590412. [Online]. Available: https://doi.org/10.1145/3583131.3590412.

[24] J. Menčík, "Latin hypercube sampling," in *Concise Reliability for Engineers*, J. Mencik, Ed., Rijeka: IntechOpen, 2016, ch. 16. DOI: 10.5772/62370. [Online]. Available: https://doi.org/10.5772/62370.

[25] A. Janković and C. Doerr, "Adaptive Landscape Analysis," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ser. GECCO '19, Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 2032–2035, ISBN: 9781450367486. DOI: 10.1145/3319619.3326905. [Online]. Available: https://doi.org/10.1145/3319619.3326905.

[26] J. Liang, B. Qu, P. Suganthan, and A. Hernández-Díaz, "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.*, Jan. 2013.

[27] J. Liang, B. Qu, and P. Suganthan, "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.*, Dec. 2013.

[28] J. Liang, B. Qu, P. Suganthan, and Q. Chen, "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.*, 2014.

[29] G. Wu, R. Mallipeddi, and P. Suganthan, "Problem Definitions and Evaluation Criteria for the CEC 2017 Competition and Special Session on Constrained Single Objective Real-Parameter Optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore.*, Oct. 2016.

[30] M. A. Muñoz and K. Smith-Miles, "Generating new space-filling test instances for continuous black-box optimization," *Evolutionary Computation*, vol. 28, pp. 379–404, Sep. 2020. DOI: 10.1162/evco_a_00262.

[31] M. Gaviano, D. E. Kvasov, D. Lera, and Y. D. Sergeyev, "Software for generation of classes of test functions with known local and global minima for global optimization," *ACM Transactions on Mathematical Software 29 (4)*, 2003.

[32] R. P. Prager, K. Dietrich, L. Schneider, *et al.*, "Neural Networks as Black-Box Benchmark Functions Optimized for Exploratory Landscape Features," in *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, 2023, pp. 129–139.

[33] D. Vermetten, F. Ye, T. Bäck, and C. Doerr, "Ma-bbob: A problem generator for black-box optimization using affine combinations and shifts," *ACM Trans. Evol. Learn. Optim.*, vol. 5, no. 1, Mar. 2025. DOI: 10.1145/3673908. [Online]. Available: https://doi.org/10.1145/3673908.

[34] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[35] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001. DOI: 10.1162/106365601750190398. [Online]. Available: https://doi.org/10.1162/106365601750190398.

[36] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[37] J. de Nobel, D. Vermetten, H. Wang, C. Doerr, and T. Bäck, "Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules," in *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*, K. Krawiec, Ed., ACM, 2021, pp. 1375–1384. DOI: 10.1145/3449726.3463167. [Online]. Available: https://doi.org/10.1145/3449726.3463167.

[38] S. van Rijn, H. Wang, M. van Leeuwen, and T. Bäck, "Evolving the structure of evolution strategies," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2016, pp. 1–8.

[39] D. Vermetten, F. Caraffini, A. V. Kononova, and T. Bäck, "Modular Differential Evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '23, Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 864–872, ISBN: 9798400701191. DOI: 10.1145/3583131.3590417. [Online]. Available: https://doi.org/10.1145/3583131.3590417.

[40] N. Hansen, A. Auger, D. Brockhoff, and T. Tusar, "Anytime performance assessment in blackbox optimization benchmarking," *IEEE Trans. Evol. Comput.*, vol. 26, no. 6, pp. 1293–1305, 2022. DOI: 10.1109/TEVC.2022.3210897. [Online]. Available: https://doi.org/10.1109/TEVC.2022.3210897.

[41] M. López-Ibáñez, D. Vermetten, J. Dréo, and C. Doerr, "Using the empirical attainment function for analyzing single-objective black-box optimization algorithms," *CoRR*, vol. abs/2404.02031, 2024. DOI: 10.48550/ARXIV.2404.02031. arXiv: 2404.02031. [Online]. Available: https://doi.org/10.48550/arXiv.2404.02031.

[42] D. Pulatov, M. Anastacio, L. Kotthoff, and H. Hoos, "Opening the black box: Automated software analysis for algorithm selection," in *Proceedings of the First International Conference on Automated Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 188, PMLR, Jul. 2022, pp. 6/1–18. [Online]. Available: https://proceedings.mlr.press/v188/pulatov22a.html.

[43] T. Eftimov, G. Popovski, D. Kocev, and P. Korošec, "Performance2vec: A Step Further in Explainable Stochastic Optimization Algorithm Performance," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '20, Cancún, Mexico: Association for Computing Machinery, 2020, pp. 193–194, ISBN: 9781450371278. DOI: 10.1145/3377929.3390020. [Online]. Available: https://doi.org/10.1145/3377929.3390020.

[44] A. Kostovska, D. Vermetten, S. Džeroski, P. Panov, T. Eftimov, and C. Doerr, "Using Knowledge Graphs for Performance Prediction of Modular Optimization Algorithms," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, Springer, 2023, pp. 253–268.

[45] A. Kostovska, C. Doerr, S. Džeroski, P. Panov, and T. Eftimov, "Geometric learning in black-box optimization: A gnn framework for algorithm performance prediction," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '25 Companion, NH Malaga Hotel, Malaga, Spain: Association for Computing Machinery, 2025, pp. 487–490, ISBN: 9798400714641. DOI: 10.1145/3712255.3726696. [Online]. Available: https://doi.org/10.1145/3712255.3726696.

[46] A. Nikolikj, A. Kostovska, D. Vermetten, C. Doerr, and T. Eftimov, "Quantifying individual and joint module impact in modular optimization frameworks," in *2024 IEEE Congress on Evolutionary Computation (CEC)*, 2024.

[47] N. van Stein, D. Vermetten, A. V. Kononova, and T. Bäck, "Explainable benchmarking for iterative optimization heuristics," *ACM Transactions on Evolutionary Learning*, vol. 5, no. 2, pp. 1–30, 2025.

[48] A. Nikolikj, R. Trajanov, G. Cenikj, P. Korošec, and T. Eftimov, "Identifying minimal set of exploratory landscape analysis features for reliable algorithm performance prediction," in *2022 IEEE Congress on Evolutionary Computation (CEC)*, 2022, pp. 1–8. DOI: 10.1109/CEC55065.2022.9870439.

[49] A. Kostovska, D. Vermetten, S. Džeroski, C. Doerr, P. Korosec, and T. Eftimov, "The Importance of Landscape Features for Performance Prediction of Modular CMA-ES Variants," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '22, Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 648–656, ISBN: 9781450392372. DOI: 10.1145/3512290.3528832. [Online]. Available: https://doi.org/10.1145/3512290.3528832.

[50] R. Trajanov, S. Dimeski, M. Popovski, P. Korošec, and T. Eftimov, "Explainable Landscape Analysis in Automated Algorithm Performance Prediction," in *Applications of Evolutionary Computation: 25th European Conference, EvoApplications 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20–22, 2022, Proceedings*, Springer, 2022, pp. 207–222.

[51] A. Nikolikj, R. Lang, P. Korošec, and T. Eftimov, "Explaining Differential Evolution Performance Through Problem Landscape Characteristics," in *International Conference on Bioinspired Optimization Methods and Their Applications*, Springer, 2022, pp. 99–113.

[52] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, pp. 197–227, 2016.

[53] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[54] N. Hollmann, S. Müller, K. Eggensperger, and F. Hutter, "TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second," in *The Eleventh International Conference on Learning Representations*, 2022.

[55] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 932–18 943, 2021.

[56] A. Kostovska, A. Jankovic, D. Vermetten, S. Džeroski, T. Eftimov, and C. Doerr, "Comparing algorithm selection approaches on black-box optimization problems," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, ser. GECCO '23 Companion, Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 495–498, ISBN: 9798400701207. DOI: 10.1145/3583133.3590697. [Online]. Available: https://doi.org/10.1145/3583133.3590697.

[57] G. Cenikj, G. Petelin, C. Doerr, P. Korošec, and T. Eftimov, "DynamoRep: Trajectory-Based Population Dynamics for Classification of Black-Box Optimization Problems," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ser. GECCO '23, Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 813–821, ISBN: 9798400701191. DOI: 10.1145/3583131.3590401. [Online]. Available: https://doi.org/10.1145/3583131.3590401.

[58] P. Korošec and T. Eftimov, "Opt2vec: A continuous optimization problem representation based on the algorithm's behavior," 2023. [Online]. Available: http://dx.doi.org/10.2139/ssrn.4316939.

[59] K. Dietrich, D. Vermetten, C. Doerr, and P. Kerschke, "Impact of Training Instance Selection on Automated Algorithm Selection Models for Numerical Black-box Optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2024, pp. 1007–1016.

[60] U. Škvorc, T. Eftimov, and P. Korošec, "A complementarity analysis of the COCO benchmark problems and artificially generated problems," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2021, pp. 215–216.

[61] G. Cenikj, R. Dieter Lang, A. Petrus Engelbrecht, C. Doerr, P. Korošec, and T. Eftimov, "SELECTOR: Selecting a Representative Benchmark Suite for Reproducible Statistical Comparison," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, 2022.

[62] A. R. Al-Roomi, "Unconstrained single-objective benchmark functions repository," *Dalhousie University, Electrical and Computer Engineering, Halifax, Nova Scotia, Canada*, 2015.

[63] P. Bennet, C. Doerr, A. Moreau, J. Rapin, F. Teytaud, and O. Teytaud, "Nevergrad: Black-box optimization platform.," in *ACM SIGEVOlution*, 2021.

[64] B. Lacroix and J. McCall, "Limitations of benchmark sets and landscape features for algorithm selection and performance prediction," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '19, Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 261–262, ISBN: 9781450367486. DOI: 10.1145/3319619.3322051. [Online]. Available: https://doi.org/10.1145/3319619.3322051.

[65] F. X. Long, D. Vermetten, A. Kononova, *et al.*, "Challenges of ELA-Guided Function Evolution Using Genetic Programming," pp. 119–130, 2023. DOI: 10.5220/0012206200003595.

[66] J. Kudela and M. Juricek, "Computational and Exploratory Landscape Analysis of the GKLS Generator," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, ser. GECCO '23 Companion, Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 443–446, ISBN: 9798400701207. DOI: 10.1145/3583133.3590653. [Online]. Available: https://doi.org/10.1145/3583133.3590653.

[67] A. Engelbrecht, P. Bosman, and K. Malan, "The influence of fitness landscape characteristics on particle swarm optimisers," *Natural Computing*, pp. 1–11, 2021.

[68] N. Belkhir, "Per Instance Algorithm Configuration for Continuous Black Box Optimization," Thesis, Université Paris-Saclay, Nov. 2017. [Online]. Available: https://hal.inria.fr/tel-01669527.

[69] P. Kerschke and H. Trautmann, "Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the r-package flacco," in *Applications in Statistical Computing: From Music Data Analysis to Industrial Quality Improvement*, N. Bauer, K. Ickstadt, K. Lübke, G. Szepannek, H. Trautmann, and M. Vichi, Eds. Springer, 2019, pp. 93–123, ISBN: 978-3-030-25147-5. DOI: 10.1007/978-3-030-25147-5_7. [Online]. Available: https://doi.org/10.1007/978-3-030-25147-5%5C_7.

[70] R. P. Prager and H. Trautmann, "Pflacco: Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems in Python," *Evolutionary Computation*, vol. 32, no. 3, pp. 211–216, Sep. 2024, ISSN: 1063-6560. DOI: 10.1162/evco_a_00341. eprint: https://direct.mit.edu/evco/article-pdf/32/3/211/2463059/evco_a_00341.pdf. [Online]. Available: https://doi.org/10.1162/evco%5C_a%5C_00341.

[71] P. Kerschke, M. Preuss, C. Hernández Castellanos, *et al.*, "Cell Mapping Techniques for Exploratory Landscape Analysis," in Jan. 2014, vol. 288, pp. 115–131, ISBN: 9783319074931. DOI: 10.1007/978-3-319-07494-8_9.

[72] M. Lunacek and D. Whitley, "The dispersion metric and the CMA evolution strategy," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 477–484.

[73] M. Preuss, "Improved topological niching for real-valued global optimization," in *Applications of Evolutionary Computation: EvoApplications 2012: EvoCOMNET, EvoCOMPLEX, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoNUM, EvoPAR, EvoRISK, EvoSTIM, and EvoSTOC, Málaga, Spain, April 11-13, 2012, Proceedings*, Springer, 2012, pp. 386–395.

[74] C. Flamm, I. L. Hofacker, P. F. Stadler, and M. T. Wolfinger, "Barrier trees of degenerate landscapes," 2002.

[75] M. A. Muñoz, M. Kirley, and S. K. Halgamuge, "Exploratory landscape analysis of continuous space optimization problems using information content," *IEEE transactions on evolutionary computation*, vol. 19, no. 1, pp. 74–87, 2014.

[76] D.-I. Seo and B.-R. Moon, "An information-theoretic analysis on the interactions of variables in combinatorial optimization problems," *Evolutionary computation*, vol. 15, no. 2, pp. 169–198, 2007.

[77] E. Weinberger, "Correlated and uncorrelated fitness landscapes and how to tell the difference," *Biological cybernetics*, vol. 63, no. 5, pp. 325–336, 1990.

[78] J. Marín, "How landscape ruggedness influences the performance of real-coded algorithms: a comparative study," *Soft Computing*, vol. 16, no. 4, pp. 683–698, 2012.

[79] Q. Renau, J. Dreo, C. Doerr, and B. Doerr, "Expressiveness and robustness of landscape features," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019, pp. 2048–2051.

[80] Q. Renau, C. Doerr, J. Dreo, and B. Doerr, "Exploratory landscape analysis is strongly sensitive to the sampling strategy," in *Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part II 16*, Springer, 2020, pp. 139–153.

[81] U. Škvorc, T. Eftimov, and P. Korošec, "The effect of sampling methods on the invariance to function transformations when using exploratory landscape analysis," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 1139–1146. DOI: 10.1109/CEC45853.2021.9504739.

[82] R. P. Prager and H. Trautmann, "Nullifying the inherent bias of non-invariant exploratory landscape analysis features," in *Applications of Evolutionary Computation*, Brno, Czech Republic: Springer-Verlag, 2023, pp. 411–425, ISBN: 978-3-031-30228-2. DOI: 10.1007/978-3-031-30229-9_27. [Online]. Available: https://doi.org/10.1007/978-3-031-30229-9%5C_27.

[83] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[84] B. van Stein, H. Wang, and T. Bäck, "Neural network design: learning from neural architecture search," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2020, pp. 1341–1349.

[85] C. Doerr, J. Dréo, and P. Kerschke, "Making a case for (Hyper-)parameter tuning as benchmark problems," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion material)*, M. López-Ibáñez, A. Auger, and T. Stützle, Eds., ACM, 2019, pp. 1755–1764. DOI: 10.1145/3319619.3326857. [Online]. Available: https://doi.org/10.1145/3319619.3326857.

[86] L. Schneider, L. Schäpermeier, R. P. Prager, B. Bischl, H. Trautmann, and P. Kerschke, "HPO× ELA: Investigating hyperparameter optimization landscapes by means of exploratory landscape analysis," in *International Conference on Parallel Problem Solving from Nature*, Springer, 2022, pp. 575–589.

[87] F. X. Long, B. van Stein, M. Frenzel, P. Krause, M. Gitterle, and T. Bäck, "Generating cheap representative functions for expensive automotive crashworthiness optimization," *ACM Trans. Evol. Learn. Optim.*, vol. 4, no. 2, Jun. 2024. DOI: 10.1145/3646554. [Online]. Available: https://doi.org/10.1145/3646554.

[88] J. Gu, Z. Wang, J. Kuen, *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[89] R. P. Prager, M. Vinzent Seiler, H. Trautmann, and P. Kerschke, "Towards feature-free automated algorithm selection for single-objective continuous black-box optimization," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021, pp. 1–8. DOI: 10.1109/SSCI50451.2021.9660174.

[90] M. V. Seiler, R. P. Prager, P. Kerschke, and H. Trautmann, "A collection of deep learning-based feature-free approaches for characterizing single-objective continuous fitness landscapes," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ser. GECCO '22, Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 657–665, ISBN: 9781450392372. DOI: 10.1145/3512290.3528834. [Online]. Available: https://doi.org/10.1145/3512290.3528834.

[91] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.

[92] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions on Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[93] D. P. Kingma, M. Welling, *et al.*, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[94] F. X. Long, B. van Stein, M. Frenzel, P. Krause, M. Gitterle, and T. Bäck, "Learning the characteristics of engineering optimization problems with applications in automotive crash," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022, pp. 1227–1236.

[95] G. Cenikj, G. Petelin, and T. Eftimov, "TransOpt: Transformer-based Representation Learning for Optimization Problem Classification," in *2023 IEEE Symposium Series on Computational Intelligence*, 2023. [Online]. Available: http://arxiv.org/abs/2311.18035.

[96] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[97] M. Seiler, U. Skvorc, C. Doerr, and H. Trautmann, "Synergies of deep and classical exploratory landscape features for automated algorithm selection," in *Proc. of Learning and Intelligent Optimization Conference (LION) 2024*, ser. Lecture Notes in Computer Science, In Press, Springer, 2024.

[98] M. Seiler, U. Škvorc, G. Cenikj, C. Doerr, and H. Trautmann, "Learned Features vs. Classical ELA on Affine BBOB Functions," in *Parallel Problem Solving from Nature – PPSN XVIII*, M. Affenzeller, S. M. Winkler, A. V. Kononova, *et al.*, Eds., Springer Nature Switzerland, 2024, pp. 137–153.

[99] G. Petelin and G. Cenikj, "Random Filter Mappings as Optimization Problem Feature Extractors," *IEEE Access*, vol. 12, pp. 143 554–143 571, 2024. DOI: 10.1109/ACCESS.2024.3468723.

[100] T. Eftimov, P. Korošec, and B. Koroušić Seljak, "A Novel Approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics," *Information Sciences*, vol. 417, pp. 186–215, 2017, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2017.07.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002002551631194X.

[101] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *International conference on machine learning*, PMLR, 2014, pp. 754–762.

[102] N. van Stein, D. Vermetten, A. V. Kononova, and T. Bäck, "Explainable benchmarking for iterative optimization heuristics," *ACM Trans. Evol. Learn. Optim.*, vol. 5, no. 2, May 2025. DOI: 10.1145/3716638. [Online]. Available: https://doi.org/10.1145/3716638.

[103] Z. Pitra, J. Repický, and M. Holena, "Landscape analysis of Gaussian process surrogates for the covariance matrix adaptation evolution strategy," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ACM, 2019, pp. 691–699. DOI: 10.1145/3321707.3321861. [Online]. Available: https://doi.org/10.1145/3321707.3321861.

[104] J. de Nobel, H. Wang, and T. Bäck, "Explorative Data Analysis of Time Series Based Algorithm Features of CMA-ES Variants," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '21, Lille, France: Association for Computing Machinery, 2021, pp. 510–518, ISBN: 9781450383509. DOI: 10.1145/3449639.3459399. [Online]. Available: https://doi.org/10.1145/3449639.3459399.

[105] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)," *Neurocomputing*, vol. 307, pp. 72–77, 2018.

[106] A. Jankovic, T. Eftimov, and C. Doerr, "Towards Feature-Based Performance Regression Using Trajectory Data," in *Proc. of Applications of Evolutionary Computation (EvoApplications 2021)*, ser. LNCS, vol. 12694, Springer, 2021, pp. 601–617. DOI: 10.1007/978-3-030-72699-7_38. [Online]. Available: https://doi.org/10.1007/978-3-030-72699-7%5C_38.

[107] A. Kostovska, A. Jankovic, D. Vermetten, *et al.*, "Per-run algorithm selection with warm-starting using trajectory-based features," in *Parallel Problem Solving from Nature (PPSN)* (LNCS), LNCS. Springer, 2022, vol. 13398, pp. 46–60. DOI: 10.1007/978-3-031-14714-2_4. [Online]. Available: https://doi.org/10.1007/978-3-031-14714-2%5C_4.

[108] D. Vermetten, H. Wang, K. Sim, and E. Hart, "To Switch or Not to Switch: Predicting the Benefit of Switching Between Algorithms Based on Trajectory Features," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, Springer, 2023, pp. 335–350.

[109] C. Benjamins, A. Jankovic, E. Raponi, K. van der Blom, M. Lindauer, and C. Doerr, "Towards Automated Design of Bayesian Optimization via Exploratory Landscape Analysis," in *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2022.

[110] G. Ochoa, S. Verel, F. Daolio, and M. Tomassini, "Local optima networks: A new model of combinatorial fitness landscapes," *Recent advances in the theory and application of fitness landscapes*, pp. 233–262, 2014.

[111] J. Adair, G. Ochoa, and K. M. Malan, "Local optima networks for continuous fitness landscapes," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019, pp. 1407–1414.

[112] D. Wales and J. Doye, "Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms," *The Journal of Physical Chemistry A*, vol. 101, Apr. 1998. DOI: 10.1021/jp970984n.

[113] P. Mitchell, G. Ochoa, and R. Chassagne, "Local Optima Networks of the Black Box Optimisation Benchmark Functions," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 2072–2080.

[114] G. Ochoa, K. M. Malan, and C. Blum, "Search trajectory networks of population-based algorithms in continuous spaces," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, Springer, 2020, pp. 70–85.

[115] Q. Renau and E. Hart, "On the utility of probing trajectories for algorithm-selection," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, Springer, 2024, pp. 98–114.

[116] G. Cenikj, G. Petelin, and T. Eftimov, "ClustOpt: A Clustering-Based Approach for Representing and Visualizing the Search Dynamics of Numerical Metaheuristic Optimization Algorithms," in *2025 IEEE Congress on Evolutionary Computation (CEC)*, 2025, pp. 1–8. DOI: 10.1109/CEC65147.2025.11043053.

[117] Y. Xian, B. Schiele, and Z. Akata, "Zero-shot learning-the good, the bad and the ugly," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4582–4591.

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

[119] R. Tanabe, "Benchmarking feature-based algorithm selection systems for black-box numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 6, pp. 1321–1335, 2022.

[120] G. Cenikj, G. Petelin, and T. Eftimov, "A cross-benchmark examination of feature-based algorithm selector generalization in single-objective numerical optimization," *Swarm and Evolutionary Computation*, vol. 87, p. 101 534, 2024, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2024.101534. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650224000725.

[121] A. Nikolikj, M. Pluhacek, C. Doerr, P. Korosec, and T. Eftimov, "Sensitivity analysis of rf+clust for leave-one-problem-out performance prediction," in *IEEE Congress on Evolutionary Computation, CEC 2023, Chicago, IL, USA, July 1-5, 2023*, IEEE, 2023, pp. 1–8, ISBN: 979-8-3503-1458-8. DOI: 10.1109/CEC53210.2023.10254146. [Online]. Available: https://doi.org/10.1109/CEC53210.2023.10254146.

[122] A. Nikolikj, S. Džeroski, M. A. Muñoz, C. Doerr, P. Korošec, and T. Eftimov, "Algorithm Instance Footprint: Separating Easily Solvable and Challenging Problem Instances," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '23, Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 529–537, ISBN: 9798400701191. DOI: 10.1145/3583131.3590424. [Online]. Available: https://doi.org/10.1145/3583131.3590424.

[123] M. A. Muñoz, M. Kirley, and S. K. Halgamuge, "A meta-learning prediction model of algorithm performance for continuous optimization problems," in *Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I 12*, Springer, 2012, pp. 226–235.

[124] G. Cenikj, G. Petelin, and T. Eftimov, "TransOptAS: Transformer-Based Algorithm Selection for Single-Objective Optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, NY, USA: Association for Computing Machinery, 2024. DOI: https://doi.org/10.1145/3638530.3654191.

[125] A. Jankovic, D. Vermetten, A. Kostovska, J. de Nobel, T. Eftimov, and C. Doerr, "Trajectory-based algorithm selection with warm-starting," in *2022 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2022, pp. 1–8.

[126] G. Cenikj, G. Petelin, C. Doerr, P. Korošec, and T. Eftimov, "Beyond Landscape Analysis: DynamoRep Features For Capturing Algorithm-Problem Interaction In Single-Objective Continuous Optimization," *Evolutionary Computation*, pp. 1–28, Mar. 2025, ISSN: 1063-6560. DOI: 10.1162/evco_a_00370. eprint: https://direct.mit.edu/evco/article-pdf/doi/10.1162/evco_a_00370/2507310/evco_a_00370.pdf. [Online]. Available: https://doi.org/10.1162/evco%5C_a%5C_00370.

[127] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2018. DOI: 10.48550/ARXIV.1810.04805. [Online]. Available: https://arxiv.org/abs/1810.04805.

[128] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, Vancouver, BC, Canada: Curran Associates Inc., 2020, ISBN: 9781713829546.

[129] Q. Wen, T. Zhou, C. Zhang, *et al.*, *Transformers in time series: A survey*, 2022. DOI: 10.48550/ARXIV.2202.07125. [Online]. Available: https://arxiv.org/abs/2202.07125.

[130] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *ArXiv*, vol. abs/1909.08053, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:202660670.

[131] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff, "COCO: A platform for comparing continuous optimizers in a black-box setting," *Optimization Methods and Software*, pp. 1–31, 2020.

[132] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106.

[133] F. X. Long, B. van Stein, M. Frenzel, P. Krause, M. Gitterle, and T. Bäck, "Learning the characteristics of engineering optimization problems with applications in automotive crash," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '22, Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 1227–1236, ISBN: 9781450392372. DOI: 10.1145/3512290.3528712. [Online]. Available: https://doi.org/10.1145/3512290.3528712.

[134] R. E. Tarjan and A. E. Trojanowski, "Finding a maximum independent set," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 537–546, 1977.

[135] P. Kerschke, M. Preuss, S. Wessing, and H. Trautmann, "Low-budget exploratory landscape analysis on multiple peaks models," Jul. 2016, pp. 229–236. DOI: 10.1145/2908812.2908845.

[136] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, Jan. 1997. DOI: 10.1023/A:1008202821328.

[137] V. Chahar, S. Katoch, and S. Chauhan, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools and Applications*, vol. 80, Feb. 2021. DOI: 10.1007/s11042-020-10139-6.

[138] H. Beyer and H. Schwefel, "Evolution strategies - a comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, Mar. 2002. DOI: 10.1023/A:1015059928466.

[139] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[140] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information Fusion*, vol. 81, pp. 84–90, 2022.

[141] G. Petelin and G. Cenikj, "The Pitfalls of Benchmarking in Algorithm Selection: What We Are Getting Wrong," in *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, NY, USA: Association for Computing Machinery, 2025, pp. 1181–1189, ISBN: 9798400714658. [Online]. Available: https://doi.org/10.1145/3712256.3726336.

[142] G. Cenikj, G. Petelin, and T. Eftimov, "Impact of Scaling in ELA Feature Calculation on Algorithm Selection Cross-Benchmark Transferability," in *2024 IEEE Congress on Evolutionary Computation (CEC)*, 2024, pp. 1–8. DOI: 10.1109/CEC60901.2024.10612032.

[143] A. Nikolikj, A. Kostovska, G. Cenikj, C. Doerr, and T. Eftimov, "Generalization ability of feature-based performance prediction models: A statistical analysis across benchmarks," in *2024 IEEE Congress on Evolutionary Computation (CEC)*, 2024, pp. 1–8. DOI: 10.1109/CEC60901.2024.10611952.

[144] B. Lacroix, L. A. Christie, and J. A. W. McCall, "Interpolated continuous optimisation problems with tunable landscape features," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '17, Berlin, Germany: Association for Computing Machinery, 2017, pp. 169–170, ISBN: 9781450349390. DOI: 10.1145/3067695.3076045. [Online]. Available: https://doi.org/10.1145/3067695.3076045.

[145] D. Vermetten, F. Ye, T. Bäck, and C. Doerr, "Ma-bbob: Many-affine combinations of bbob functions for evaluating automl approaches in noiseless numerical black-box optimization contexts," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ser. GECCO '23, Lisbon, Portugal: Association for Computing Machinery, Jun. 2023, pp. 813–821.

[146] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[147] B. Stein, *Doe2vec*, https://pypi.org/project/doe2vec/, Accessed: 2024-10-02, 2024.

[148] K. Dietrich, D. Vermetten, C. Doerr, and P. Kerschke, "Impact of training instance selection on automated algorithm selection models for numerical black-box optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '24, ACM, Jul. 2024, pp. 1007–1016. DOI: 10.1145/3638529.3654100. [Online]. Available: http://dx.doi.org/10.1145/3638529.3654100.

[149] Q. Renau, C. Doerr, J. Dreo, and B. Doerr, "Exploratory landscape analysis is strongly sensitive to the sampling strategy," in *Proc. of Parallel Problem Solving from Nature (PPSN)*, Sep. 2020, pp. 139–153, ISBN: 978-3-030-58115-2.

[150] A. Jankovic, T. Eftimov, and C. Doerr, "Towards feature-based performance regression using trajectory data," in *Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, April 7–9, 2021, Proceedings 24*, Springer, 2021, pp. 601–617.

[151] B. Derbel, A. Liefooghe, S. Verel, H. Aguirre, and K. Tanaka, "New Features for Continuous Exploratory Landscape Analysis based on the SOO Tree," in *FOGA 2019 - 15th ACM/SIGEVO Workshop on Foundations of Genetic Algorithms*, ffhal-02282986, Potsdam, Germany, Aug. 2019, pp. 72–86.

[152] G. Ochoa, K. M. Malan, and C. Blum, "Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics," *Applied Soft Computing*, vol. 109, p. 107 492, 2021.

[153] H. Gholamalinezhad and H. Khosravi, "Pooling methods in deep neural networks, a review," *arXiv preprint arXiv:2009.07485*, 2020.

[154] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 260–13 271, 2020.

[155] E. T. McGonigle, R. Killick, and M. A. Nunes, "Modelling time-varying first and second-order structure of time series via wavelets and differencing," *Electronic Journal of Statistics*, vol. 16, no. 2, pp. 4398–4448, 2022. DOI: 10.1214/22-EJS2044. [Online]. Available: https://doi.org/10.1214/22-EJS2044.

[156] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer, "Per instance algorithm configuration of CMA-ES with limited budget," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ACM, 2017, pp. 681–688. DOI: 10.1145/3071178.3071343. [Online]. Available: https://doi.org/10.1145/3071178.3071343.

[157] N. v. Stein and T. Bäck, "Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics," *Trans. Evol. Comp*, vol. 29, no. 2, pp. 331–345, Nov. 2024, ISSN: 1089-778X. DOI: 10.1109/TEVC.2024.3497793. [Online]. Available: https://doi.org/10.1109/TEVC.2024.3497793.

[158] W. Li, N. van Stein, T. Bäck, and E. Raponi, *Llamea-bo: A large language model evolutionary algorithm for automatically generating bayesian optimization algorithms*, 2025. arXiv: 2505.21034 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2505.21034.

[159] M. Pluhacek, J. Kovac, A. Viktorin, P. Janku, T. Kadavy, and R. Senkerik, "Using llm for automatic evolvement of metaheuristics from swarm algorithm soma," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '24 Companion, Melbourne, VIC, Australia: Association for Computing Machinery, 2024, pp. 2018–2022, ISBN: 9798400704956. DOI: 10.1145/3638530.3664181. [Online]. Available: https://doi.org/10.1145/3638530.3664181.

# Bibliography

## Publications Related to the Thesis

### Journal Articles

G. Petelin and G. Cenikj, "Random Filter Mappings as Optimization Problem Feature Extractors," *IEEE Access*, vol. 12, pp. 143 554–143 571, 2024. DOI: 10.1109/ACCESS.2024.3468723.

G. Cenikj, G. Petelin, M. Seiler, N. Cenikj, and T. Eftimov, "Landscape features in single-objective continuous optimization: Have we hit a wall in algorithm selection generalization?" *Swarm and Evolutionary Computation*, vol. 94, p. 101 894, 2025, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2025.101894. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650225000525.

G. Cenikj, G. Petelin, and T. Eftimov, "A cross-benchmark examination of feature-based algorithm selector generalization in single-objective numerical optimization," *Swarm and Evolutionary Computation*, vol. 87, p. 101 534, 2024, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2024.101534. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650224000725.

G. Petelin, G. Cenikj, and T. Eftimov, "TinyTLA: Topological landscape analysis for optimization problem classification in a limited sample setting," *Swarm and Evolutionary Computation*, vol. 84, p. 101 448, 2024, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2023.101448. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650223002201.

G. Cenikj, G. Petelin, C. Doerr, P. Korošec, and T. Eftimov, "Beyond Landscape Analysis: DynamoRep Features For Capturing Algorithm-Problem Interaction In Single-Objective Continuous Optimization," *Evolutionary Computation*, pp. 1–28, Mar. 2025, ISSN: 1063-6560. DOI: 10.1162/evco_a_00370. eprint: https://direct.mit.edu/evco/article-pdf/doi/10.1162/evco_a_00370/2507310/evco_a_00370.pdf. [Online]. Available: https://doi.org/10.1162/evco%5C_a%5C_00370.

### Conference Paper

M. Seiler, U. Škvorc, G. Cenikj, C. Doerr, and H. Trautmann, "Learned Features vs. Classical ELA on Affine BBOB Functions," in *Parallel Problem Solving from Nature – PPSN XVIII*, M. Affenzeller, S. M. Winkler, A. V. Kononova, *et al.*, Eds., Springer Nature Switzerland, 2024, pp. 137–153.

A. Nikolikj, A. Kostovska, G. Cenikj, C. Doerr, and T. Eftimov, "Generalization ability of feature-based performance prediction models: A statistical analysis across benchmarks," in *2024 IEEE Congress on Evolutionary Computation (CEC)*, 2024, pp. 1–8. DOI: 10.1109/CEC60901.2024.10611952.

G. Cenikj, R. Dieter Lang, A. Petrus Engelbrecht, C. Doerr, P. Korošec, and T. Eftimov, "SELECTOR: Selecting a Representative Benchmark Suite for Reproducible Statistical Comparison," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, 2022.

G. Petelin, G. Cenikj, and T. Eftimov, "TLA: Topological Landscape Analysis for Single-Objective Continuous Optimization Problem Instances," in *Proceedings of IEEE Symposium Series On Computational Intelligence*, 2022, pp. 1698–1705. DOI: 10.1109/SSCI51031.2022.10022126.

G. Cenikj, G. Petelin, C. Doerr, P. Korošec, and T. Eftimov, "DynamoRep: Trajectory-Based Population Dynamics for Classification of Black-Box Optimization Problems," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ser. GECCO '23, Lisbon, Portugal: Association for Computing Machinery, 2023, pp. 813–821, ISBN: 9798400701191. DOI: 10.1145/3583131.3590401. [Online]. Available: https://doi.org/10.1145/3583131.3590401.

G. Petelin and G. Cenikj, "The Pitfalls of Benchmarking in Algorithm Selection: What We Are Getting Wrong," in *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, NY, USA: Association for Computing Machinery, 2025, pp. 1181–1189, ISBN: 9798400714658. [Online]. Available: https://doi.org/10.1145/3712256.3726336.

G. Cenikj, G. Petelin, and T. Eftimov, "ClustOpt: A Clustering-Based Approach for Representing and Visualizing the Search Dynamics of Numerical Metaheuristic Optimization Algorithms," in *2025 IEEE Congress on Evolutionary Computation (CEC)*, 2025, pp. 1–8. DOI: 10.1109/CEC65147.2025.11043053.

G. Petelin and G. Cenikj, "How Far Out of Distribution Can We Go With ELA Features and Still Be Able to Rank Algorithms?" In *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2023, pp. 341–346.

G. Cenikj, G. Petelin, and T. Eftimov, "Impact of Scaling in ELA Feature Calculation on Algorithm Selection Cross-Benchmark Transferability," in *2024 IEEE Congress on Evolutionary Computation (CEC)*, 2024, pp. 1–8. DOI: 10.1109/CEC60901.2024.10612032.

## Companion Paper

G. Cenikj, G. Petelin, and T. Eftimov, "TransOptAS: Transformer-Based Algorithm Selection for Single-Objective Optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, NY, USA: Association for Computing Machinery, 2024. DOI: https://doi.org/10.1145/3638530.3654191.

# Biography

Gjorgjina Cenikj was born in 1998 in Skopje, Macedonia. In 2020, she obtained her Bachelor degree in the area of Computer Science and Engineering from the Faculty of Computer Science and Engineering at the Ss. Cyril and Methodius University in Skopje, Macedonia. She completed her master studies in the area of Information and Communication Technologies at the Jožef Stefan International Postgraduate School in Ljubljana, Slovenia. Her master thesis targeted the development of Information Extraction methodologies for the domain of food and nutrition. During her master studies, she was the recipient of the Ad Futura scholarship for the Western Balkans.

In the fall of 2022, she started her PhD studies at the same institution, under the supervision of Dr. Tome Eftimov and co-supervision of Prof. Dr. Barbara Koroušić Seljak. She is currently a Young Researcher at the Jožef Stefan Institute in Ljubljana, Slovenia, working at the intersection of machine learning and optimization.

Her main research interests include machine learning, optimization, representation learning, deep learning, and natural language processing. She has published her work in international journals and has presented it at several conferences.