REPRESENTING AND EXPLOITING BENCHMARKING DATA FOR OPTIMISATION AND LEARNING

Ana Kostovska

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Asst. Prof. Panče Panov, Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
Co-Supervisor: Prof. Dr. Sašo Džeroski, Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia
Co-Supervisor: Asst. Prof. Tome Eftimov, Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:
Prof. Dr. Peter Korošec, Chair, Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia
Dr. Carola Doerr, Member, Sorbonne Université, CNRS, Paris, France
Prof. Dr. Larisa Soldatova, Member, Goldsmiths, University of London, The United Kingdom

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Ana Kostovska

REPRESENTING AND EXPLOITING BENCHMARKING DATA FOR OPTIMISATION AND LEARNING

Doctoral Dissertation

PREDSTAVITEV IN UPORABA PODATKOV IZ PRIMERJALNIH ŠTUDIJ ZA OPTIMIZACIJO IN UČENJE

Doktorska disertacija

Supervisor: Asst. Prof. Panče Panov

Co-Supervisor: Prof. Dr. Sašo Džeroski

Co-Supervisor: Asst. Prof. Tome Eftimov

Ljubljana, Slovenia, December 2024

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisors. I am thankful to Asst. Prof. Panče Panov for providing me with his guidance throughout my PhD journey. I extend my sincere thanks to Prof. Dr. Sašo Džeroski for giving me the opportunity to embark on this academic path. To Asst. Prof. Tome Eftimov, whose unwavering support has been instrumental during my PhD studies, thank you for introducing me to the fascinating world of research and science and for making every step of this process more enjoyable and fulfilling. Without your guidance and encouragement, none of this would have been possible.

I would also like to express my gratitude to the members of the evaluation board – Prof. Dr. Peter Korošec, Dr. Carola Doerr, and Prof. Dr. Larisa Soldatova – for their time and effort in reviewing my work and providing valuable feedback.

I am also deeply thankful to my colleagues at the Department of Knowledge Technologies for their support and collaboration throughout my PhD. Additionally, I extend my appreciation to the Computer Systems Department, where I have always felt welcomed and part of their team.

I am grateful to the SPECIES Society for supporting my research visit to LIP6, which was a pivotal experience in my academic development. I would like to thank the entire RO team at LIP6 for their warm welcome. In particular, I am deeply thankful to Dr. Carola Doerr for being an exceptional host and providing me with the opportunity to broaden my scientific knowledge.

My thanks also goes to Diederick Vermetten for his constant availability to address my questions on black-box optimization. Your support has been invaluable.

My heartfelt thanks go to Gordana Ispirova, Gjorgjina Cenikj, Ana Nikolikj, and Tome Eftimov, who have been like a second family to me throughout these years, sharing countless memorable moments along the way.

Finally, I wish to thank all my friends and family, who, though not named individually here, have been the most important part of this journey. Their encouragement, patience, and love have carried me through the challenges and triumphs of this PhD.

Abstract

The rapid advancements in Machine Learning (ML) and Black-Box Optimization (BBO) have led to an increased reliance on benchmarking data for evaluating and comparing algorithms across diverse domain tasks. However, the effective exploitation of this data is hindered by challenges such as syntactic variability, semantic ambiguity, and lack of standardization. In this dissertation, we address these challenges by advocating for formal semantic representation of benchmarking data through the use of ontologies. By providing standardized vocabularies and ontologies, we improve knowledge sharing and promote data interoperability across studies in ML and BBO.

In the ML domain, focusing on multi-label classification (MLC), we design an ontology-based framework for semantic annotation of benchmarking data, facilitating the creation of MLCBench – a semantic catalog that enhances data accessibility and reusability. In the BBO domain, we introduce the OPTION (OPTImization algorithm benchmarking ONtology) ontology to formally represent benchmarking data, including performance data, algorithm metadata, and problem landscapes. This ontology enables the automatic integration and interoperability of knowledge and data from diverse benchmarking studies.

Building upon the semantically annotated benchmarking data, we conduct various empirical studies, including tasks such as algorithm performance prediction and automated algorithm selection (AAS). In the MLC domain, a data-driven AAS pipeline is proposed to exploit this MLC benchmarking data. We evaluate the predictive power of dataset meta-features for AAS and explore various ML approaches – including regression, classification, and pairwise methods – to identify the most effective one.

In the BBO domain, we exploit benchmarking data about modular BBO algorithms to conduct a comprehensive analysis of how individual algorithm modules influence overall performance. We develop algorithm representations derived from performance and feature importance values, effectively linking algorithm behavior to problem landscape features. Using these representations, we also relate module configurations and performance, providing deeper insights into the impact of different modules on algorithm performance.

Furthermore, the semantically annotated benchmarking data on modular BBO optimization algorithms is used as a backbone for creating various knowledge graphs (KGs). The KGs are then examined for their predictive power in algorithm performance prediction. By applying scoring-based KG embedding methods and graph neural networks, we predict algorithm performance in transductive and inductive setups, respectively.

Overall, the contributions of this dissertation include the development of ontology-based frameworks for managing benchmarking data in the ML and BBO domains, the creation of semantic data catalogs, and novel methodologies for algorithm selection and performance prediction. By addressing challenges in representation and exploitation, this work advances both ML and BBO. It provides tools for improved data management and algorithm selection, as well as insights into algorithm behavior.

Povzetek

Nagel razvoj strojnega učenja in optimizacije črnih skrinjic je privedel do večje odvisnosti od primerjalnih podatkov za vrednotenje in primerjavo algoritmov na različnih področjih, vendar pa učinkovito izkoriščanje teh podatkov otežujejo izzivi, kot so sintaktična raznolikost, semantična dvoumnost in pomanjkanje standardizacije. Pričujoča disertacija se ukvarja s temi izzivi in zagovarja formalno semantično predstavitev primerjalnih podatkov z uporabo ontologij. Uporaba ontologij izboljšuje deljenje znanja in spodbuja interoperabilnost podatkov med raziskavami v strojnem učenju in optimizaciji črnih skrinjic.

Na področju strojnega učenja, s poudarkom na nalogi večoznačne klasifikacije, razvijamo okvir, ki temelji na ontologijah za semantično označevanje primerjalnih podatkov, kar omogoča oblikovanje MLCBench, semantičnega kataloga za izboljšanje dostopnosti in uporabnosti podatkov. Na področju optimizacije črnih skrinjic uvajamo ontologijo OPTION (OPTImization algorithm benchmarking ONtology), ki formalno predstavlja primerjalne podatke, vključno s podatki o uspešnosti, metapodatki algoritmov in značilnostih problemov. Tovrstna ontologija omogoča integracijo ter interoperabilnost znanja in podatkov iz različnih študij. Na podlagi semantično označenih primerjalnih podatkov izvajamo različne empirične študije, vključno z nalogami, kot sta napovedovanje uspešnosti algoritmov in samodejna izbira algoritmov.

Na področju večoznačne klasifikacije predlagamo delotok samodejne izbire algoritmov, ki uporablja te primerjalne podatke, temelječe na večoznačni klasifikaciji. Ocenjujemo napovedno moč metaznačilk iz podatkovnih množic za nalogo samodejne izbire algoritmov ter raziskujemo različne pristope strojnega učenja – vključno z regresijo, klasifikacijo in metodami parnega ujemanja –, da bi identificirali najučinkovitejši pristop.

Na področju optimizacije črnih skrinjic izkoriščamo primerjalne podatke modularnih algoritmov za celovito analizo, kako posamezni moduli algoritmov vplivajo na skupno uspešnost. Razvijamo metapredstavitve, ki temeljijo na vrednostih uspešnosti in pomenu značilnostmi, s čimer učinkovito povezujemo obnašanje algoritmov z značilnostmi problemov. Z uporabo teh metapredstavitev prav tako napovedujemo konfiguracije modulov, kar omogoča globlji vpogled v vpliv posameznih modulov na uspešnost algoritmov.

Poleg tega uporabljamo semantično označene primerjalne podatke modularnih algoritmov, in sicer kot osnovo za ustvarjanje različnih grafov znanja, ki jih nato uporabljamo za preučevanje njihove napovedne moči za napovedovanje uspešnosti algoritmov. S pristopoma vstavitve grafov znanja na osnovi točkovanja in grafovskih nevronskih mrež napovedujemo uspešnost algoritmov v transduktivnih in induktivnih scenarijih.

Prispevki pričujoče disertacije vključujejo razvoj ontoloških okvirov za upravljanje primerjalnih podatkov na področjih strojnega učenja in optimizacije črnih skrinjic, ustvarjanje semantičnih podatkovnih katalogov ter nove metodologije za izbiro in napovedovanje uspešnosti algoritmov. Z obravnavo vidikov predstavljanja in uporabe podatkov iz primerjalnih študij disertacija izpopolnjuje področji strojnega učenja in optimizacije črnih skrinjic ter zagotavlja orodja, ki omogočajo boljše upravljanje podatkov, izboljšano izbiro algoritmov in globlje razumevanje njihovega delovanja.

Contents

List of Figures

List of Tables

Abbreviations

AI	... Artificial Intelligence
AAS	... Automated Algorithm Selection
ANNs	... Artificial Neural Networks
AS	... Algorithm Selection
AutoML	... Automated Machine Learning
BBO	... Black Box Optimization
BBOA	... Black-Box Optimization Algorithm
CMA-ES	... Covariance Matrix Adaptation Evolution Strategies
DE	... Differential Evolution
DL	... Description Logic
DSC	... Deep Statistical Comparison
DNNs	... Deep Neural Networks
EC	... Evolutionary Computation
ELA	... Exploratory Landscape Analysis
FAIR	... Findable, Accessible, Interoperable, Reusable
IAO	... Information Artifact Ontology
KB	... Knowledge Base
KGC	... Knowledge Graph Completion
KGE	... Knowledge Graph Embedding
KGR	... Knowledge Graph Reasoning
KGRL	... Knowledge Graph Representation Learning
KG	... Knowledge Graph
KR	... Knowledge Representation
LHS	... Latin Hypercube Sampling
ML	... Machine Learning
MLC	... Multi-label Classification
MtL	... Meta-Learning
OBI	... Ontology of Biomedical Investigations
OntoDM-core	... The Ontology of Core Data Mining Entities
OntoDT	... The Generic Ontology of Datatypes
OWL	... Web Ontology Language
RDF	... Resource Description Framework
RDFS	... RDF Schema
RF	... Random Forests
RO	... Relations Ontology
SBS	... Single Best Solver
TRUST	... Transparency, Responsibility, User focus, Sustainability, and Technology
VBS	... Virtual Best Solver
W3C	... World Wide Web Consortium

Chapter 1

Introduction

1.1 Study Domains

In recent years, the growing demand for effective and efficient solutions to complex problems across various domains has led to a significant increase in the use of computational techniques. Two scientific domains that have emerged as crucial in addressing such challenges are Machine Learning (ML) and Black-Box Optimization (BBO).

ML has revolutionized the way we analyze data and build predictive models, providing advanced methods to uncover patterns and relationships in complex datasets. Its applications span across diverse domains such as healthcare, medical diagnostics, environmental modeling, financial forecasting, autonomous systems, and personalized medicine [1]. On the other hand, BBO specializes in solving optimization problems for which the internal workings of the objective function are unknown or cannot be explicitly modeled. This approach is particularly valuable in engineering design, hyperparameter tuning in ML models, and operations research, where the goal is to identify the best solution based solely on the input-output behavior of the system [2].

Both domains, despite their distinct focus, share a common objective: to leverage data and computational strategies to tackle complex, often high-dimensional problems that are otherwise intractable using traditional methods.

This dissertation focuses on two subfields within the broader domains of ML and BBO:

- **Multi-Label Classification (MLC)** in ML, a challenging predictive modeling task where instances may have multiple labels simultaneously. This task is relevant in applications such as text categorization, image tagging, and bioinformatics, where outputs are inherently interdependent [3].
- **Single-Objective Numerical Black-Box Optimization** in BBO, a foundational area in optimization that aims to minimize or maximize a single objective function without explicit knowledge of its analytical form.

1.2 The Role and Types of Benchmarking Data

Central to the progress in both ML and BBO is the generation and use of benchmarking data. **Benchmarking data** is a product of executing controlled computational experiments to evaluate and compare the performance of algorithms on a variety of tasks. It typically includes **performance data**, which captures quantitative measures of algorithm effectiveness and/or efficiency; **algorithm metadata**, which provides information about configurations and hyperparameter settings; and **problem metadata**, describing the characteristics of the tasks or problems being solved. Additionally, benchmarking data often

includes **experimental setup information**, detailing the conditions under which the experiments were conducted, and the evaluation protocols being used, as well as **provenance information** documenting the data generation process, such as tools, platforms, and contributors. In ML, benchmarking data is the foundation of meta-learning, where past performance data is used to improve future learning processes – essentially, learning how to learn [4]. In BBO, benchmarking data provides critical insights into the behavior of optimization algorithms across problem landscapes, informing algorithm design and selection [5].

1.3 Challenges in Exploiting Benchmarking Data

Despite the importance of benchmarking data, it is often underutilized due to three significant challenges:

1. **Syntactic Variability:** Benchmarking data is stored in diverse formats across different studies and platforms. This lack of uniformity and clarity of representation creates barriers to integration and interoperability, complicating the process of drawing meaningful insights from the data.

2. **Semantic Ambiguity:** Inconsistent labeling, insufficient metadata, and varying semantic interpretations of performance metrics hinder the clarity and reusability of benchmarking data. Researchers often struggle to understand or reuse data due to these ambiguities.

3. **Lack of Standardization:** The absence of standardized protocols for data representation and evaluation metrics makes it difficult to compare algorithm performance or replicate experiments. This fragmentation limits collaborative advancements and inhibits the scalability of research efforts.

Addressing these challenges requires a paradigm shift in how benchmarking data is represented, stored, and exploited. This dissertation argues that formal semantic representation is key to unlocking the full potential for exploitation of benchmarking data. By leveraging ontologies – semantic frameworks that formalize domain knowledge – it is possible to standardize domain knowledge and data representation, enhance querying capabilities, and promote interoperability across studies. Ontologies are not only essential for overcoming the intrinsic challenges of data variability and ambiguity but also critical for aligning with state-of-the-art community data management guidelines, such as the FAIR data principles [6], which emphasize Findability, Accessibility, Interoperability, and Reusability of data. By providing a standardized vocabulary and formal structure, ontologies ensure that benchmarking data is machine-readable, reusable, and interoperable across diverse research contexts. Ontologies also facilitate the creation of knowledge graphs (KGs), which integrate data and metadata into rich, relational structures, unlocking new opportunities for exploration.

1.4 Exploiting Benchmarking Data

The rich benchmarking data generated in the ML and BBO domains offers significant exploitation potential for advanced analyses. Two key tasks of exploration include:

(i) **Automated Algorithm Selection (AAS) [7]:** This task involves determining the most appropriate algorithm for a given task by using benchmarking data. By uncovering performance patterns and analyzing relationships between problem characteristics and algorithm configurations, AS facilitates more informed and effective decision-making.

(ii) **Algorithm Performance Prediction [8]:** This task focuses on building models to estimate algorithm performance for specific problems, e.g., BBO tasks or MLC datasets based on benchmarking data. Typically, these models rely heavily on problem landscape data, leveraging features that describe the problem to forecast outcomes, i.e., performance figures for BBO or MLC algorithms.

However, traditional approaches to AS and performance prediction often neglect crucial algorithm-specific details, thereby limiting their scope and potential impact. A promising alternative lies in the use of **knowledge graphs (KGs)**, which are uniquely suited to capturing relational data between entities such as algorithms, problems, and their respective characteristics. Unlike conventional methods, KGs seamlessly integrate algorithm-specific details, problem landscape data, and performance data into a cohesive, interconnected framework, offering a richer and more comprehensive representation of benchmarking data.

1.5 Problem Formulation

The overall structure of this doctoral work can be viewed along two orthogonal dimensions: one based on the study domain (ML vs. BBO) and another based on representing vs. exploiting benchmarking data. Within the ML domain, our focus is on predictive modeling, specifically on the MLC task, while in BBO we focus on single-objective continuous BBO. By studying the relevant literature and resources, we have identified several open issues to further explore in the doctoral dissertation:

1. **Lack of adequate representation of benchmarking data in the domains of ML and BBO.** Adequate formal semantic representation of benchmarking data in both study domains are currently lacking. Although some efforts have been made toward creating ontologies, they fall short of providing the necessary vocabularies to encompass all aspects of benchmarking studies. As a result, the ontological formalization of domain knowledge in both ML and BBO domains remains incomplete. Further work is required to fully establish a comprehensive and precise representation of domain knowledge in these areas in order to create machine-actionable semantically rich data and metadata.

2. **Exploitation of benchmarking data in the domains of ML and BBO is limited.** When it comes to the exploitation of the benchmarking data, there are several open issues we aim to tackle in this work in both study domains:

    - AAS in the realm of MLC remains largely unexplored, leaving a significant gap in understanding what are the most suitable algorithms for MLC tasks. Furthermore, there are no comprehensive guidelines regarding the effective characterization and description of MLC datasets that can be used for MLC AAS. For instance, a notable contribution by Moyano et al. [9] introduced a set of MLC dataset metafeatures that capture the various measurable properties of MLC tasks. These metafeatures, among others, include the number of descriptive attributes, data instances, labels, and statistical properties of the descriptive attributes. While these meta-representations are crucial for leveraging MLC datasets in predictive modeling tasks, their application in AS remains unexplored. Additionally, AAS in machine learning can be approached using various techniques, including regression models, classifiers, pairwise variants, and single- or multi-output methods. However, the impact of the ML technique chosen at the meta-level on AAS performance is not well understood.

- Relating algorithm behavior with algorithm and problem properties is not well explored in a systematic way. In BBO, we focus on benchmarking data about modular optimization algorithms. The idea behind modular optimization algorithm frameworks is to break down the algorithm into smaller components that can be easily modified and combined to create new algorithms. By using a modular approach, researchers can better understand how individual components contribute to the algorithm's overall performance and identify areas for improvement. Comprehensive empirical studies that thoroughly examine the impact of different algorithm operators or algorithm modules on performance, considering the landscape properties of the problems, are still missing. Understanding how different algorithm operators interact with problem characteristics is crucial for enhancing algorithm performance.
- The predictive power of KG derived from benchmarking data as semantic representations has not been exploited in various ML and BBO tasks. Leveraging KGs in this context could provide valuable insights into algorithm behaviour and improve the performance of algorithm performance predictions models.

In the proposed dissertation, we will focus on addressing the above identified open issues. This will contribute to a more robust and comprehensive representation and exploitation of benchmarking data in both MLC and BBO domains, leading to advances in AAS, empirical studies that analyse algorithm behaviour, and the utilization of KGs for algorithm performance predictions.

1.6 Purpose of the Dissertation

The main purpose of this dissertation is to **develop methods and resources for representing and exploiting benchmarking data for optimization and learning**. In terms of representation, our focus is on creating a formal ontology-based framework specifically tailored for benchmarking data in these two domains. We also aim to exploit the benchmarking data to address various learning tasks, such as algorithm performance prediction and AAS.

1.7 Goals of the Dissertation

The goals of this dissertation are as follows:

G1. Design an ontology-based scheme for semantic annotation of benchmarking data in MLC, develop data annotation pipelines, annotate benchmarking data, and develop software for easy access, querying, and reuse of the annotated MLC benchmarking data.

G2. Develop an ontology for formal representation of benchmarking data and knowledge in the domain of BBO, develop data annotation pipelines, annotate benchmarking data, and develop software for easy access, querying, and reuse of the annotated BBO benchmarking data.

G3. Develop AAS pipelines for MLC, including training various classification and regression models, constructing the selector pipeline, and incorporating an explainability layer to identify crucial dataset characteristics for algorithm performance prediction.

G4. Develop ML-based pipelines for: (1) relating modular BBO algorithm components to algorithm behavior, (2) investigation of the importance of problem landscape characteristics in modular BBO algorithm performance prediction, and (3) prediction of the algorithm's modular configuration.

G5. Evaluate the predictive capabilities of KGs for modular BBO algorithm performance prediction.

1.7.1 Research questions

Here we list the research questions we will consider in the proposed dissertation. Two research questions (R1-R2) are related to the representation aspects of the work and three research questions (R3-R5) are related to the exploitation aspect of the work.

R1. Can an MLC ontology-based semantic annotation scheme be designed and applied to annotate MLC benchmarking data to enable easy data accessibility, improved querying capabilities, increased reusability, and support for automated data integration and domain knowledge sharing?

R2. Can a BBO benchmarking ontology be designed and applied to annotate BBO benchmark data to enhance data accessibility, querying capabilities, reusability, and enable automated data integration and domain knowledge sharing?

R3. Does the development of a data-driven AAS pipeline for multi-label classification (MLC) lead to better AS practices by leveraging dataset-specific characteristics, and how does it compare to static approaches using a single algorithm across all datasets?

R4. Can a systematic empirical analysis of modular BBO algorithm behavior, combined with algorithm and problem characterization, improve our understanding of the impact of algorithm modules and problem landscape characteristics on algorithm performance?

R5. Are KGs as semantic data representations effective for predicting the performance of modular BBO algorithms?

1.7.2 Scientific contributions

The dissertation aims to make scientific contributions to the fields of ML and BBO by addressing research gaps related to the lack of adequate representations and limited exploration of benchmarking data in these domains. The novelty and originality of this dissertation stem from the development of ontology-based frameworks for the formal representation of benchmarking data, an aspect that has not been adequately considered before. Additionally, we propose novel approaches for exploiting and learning from benchmarking data for various learning tasks, such as AAS and algorithm performance prediction.

By employing the ontology-based framework for formal data representation, our methodology strives to enhance data reusability and promote knowledge sharing among practitioners in the field. This approach is expected to facilitate improved collaboration among domain experts and encourage the reuse of research results, thereby paving the way for advancements in the domains of ML and BBO.

The development of pipelines for automated algorithm performance prediction and selection will enable industry professionals to make informed decisions, optimize resources, and achieve improved outcomes in various sectors relying on ML and BBO (e.g., healthcare and finance). This approach leverages benchmarking data to gain a better understanding of

algorithm behavior, ultimately enhancing decision-making processes and driving successful algorithm deployment.

To summarize, the dissertation will make the following contributions to science:

C1. An ontology-based framework together with a semantic annotation schema and software, that enables formal representation, annotation, and querying of MLC benchmarking data.

C2. Semantically annotated MLC benchmarking data that promotes data reusability, interoperability, and knowledge sharing enabling its further exploitation in different learning tasks.

C3. An ontology-based framework, including an ontological conceptualization and software, that enables formal representation, annotation, and querying of BBO benchmarking data.

C4. Semantically annotated BBO benchmarking data that integrates data from different benchmarking studies and promotes data reusability, interoperability, and knowledge sharing enabling its further exploitation in different learning tasks.

C5. A novel pipeline for AAS in MLC.

C6. A novel approach and extensive empirical evaluation for understanding modular BBO algorithm behavior.

C7. A novel KG-based approach to BBO modular algorithm performance prediction.

1.8 Methodology

To achieve the goals outlined above, we have proposed the following methodology. One part focuses on representing benchmarking data, while the other focuses on exploiting the benchmarking data in different predictive modeling tasks.

1.8.1 Representation of Benchmarking Data

In the representation component of this dissertation, we adopted a hybrid knowledge representation methodology that combines both top-down and bottom-up strategies to develop formal, semantic representations of knowledge and data in the study domains.

First, we conducted an in-depth overview of the study domains (MLC and BBO), gaining familiarity with the domain knowledge and identifying the key entities and concepts central to benchmarking data. This top-down approach involves understanding the structure and relationships inherent to the domain, allowing us to define a conceptual framework for its representation.

Simultaneously, we took a bottom-up approach to identify the types of data that need to be modeled and stored. This includes the identification of elements such as performance metrics, problem landscapes, experimental setups, provenance information, and algorithm meta-representations. By taking this hybrid approach, we ensured that the representation of domain knowledge and data was both conceptually sound and grounded in real-world benchmarking practices.

The next phase focused on designing and developing semantic models, including schemas and ontologies, specifically tailored to the requirements of the study domains (ML and BBO) identified in the previous step. During the ontology design process, we adhered to

best practices in ontology engineering, such as the OBO Foundry principles [10], to ensure compatibility and interoperability with other external ontologies that follow the same design principles. These semantic models form the foundation for formalizing domain knowledge and enabling effective data representation and integration.

Once the semantic models were established, we carried out the process of semantic data annotation. This step involves linking benchmarking data, metadata, and domain knowledge to the structured entities and relationships defined in the semantic models. Through a process of extraction, transformation, and loading (ETL), we created knowledge bases (KB) that integrate the annotated data, ensuring that raw and heterogeneous information is harmonized into a unified and semantically enriched format.

As a final step, we implemented the developed semantic models into practical systems and tools. This includes creating systems for storing and querying semantic annotations. Additionally, we developed software components, including REST APIs and graphical interfaces, to make ontologies easier to use. These tools simplify interaction with the semantic models by abstracting complex querying processes. Our aim is to make these solutions and developed resources accessible to practitioners in the fields of ML and BBO, enabling them to efficiently reuse and leverage benchmarking resources.

1.8.2 Exploitation of Benchmarking Data

After creating semantic benchmarking resources and developing the necessary tools, our efforts shifted toward leveraging these resources to explore and analyze benchmarking data. Here, we summarise the methodology employed for exploitation.

1.8.2.1 Automated algorithm selection in multi-label classification

In the MLC domain, we focused on developing a data-driven, AAS pipeline. We utilized existing meta-representations of MLC datasets [9], available in the MLC knowledge base of semantically annotated benchmarking data we have created. These meta-representations capture measurable properties like the number of attributes, instances, labels, and statistical characteristics. Using these meta-features, we assessed their predictive power for algorithm selection.

We employed several feature-based supervised ML approaches, including regression, classification, and pairwise methods, in both single-output and multi-output configurations. Regression models predicted each algorithm's performance on a dataset, selecting the algorithm with the best predicted outcome. Classification framed AAS as a multi-class problem, directly predicting the best algorithm based on dataset meta-features.

Pairwise methods predicted the relative performance of algorithm pairs. Pairwise regression estimated performance differences, while pairwise classification determined which algorithm performed better in each pair. In this context, final AAS aggregated pairwise predictions to identify the algorithm with the most "wins".

To improve interpretability, we integrated an explainability layer using SHAP (SHapley Additive exPlanations) values [11], enabling us to assess the importance of each meta-feature in the models' predictions both globally and per instance.

1.8.2.2 Analysis of modular optimization algorithms in black-box optimization

In the BBO domain, we developed a data-driven approach to assess how individual modules within modular optimization algorithms influence the overall algorithm performance. Focusing on two modular frameworks, we generated algorithm meta-representations for

each algorithm variant—specific combinations of modules—capturing their performance profiles across various problem classes.

We created two types of meta-representations. The first, performance-based meta-representations, involved summarizing each algorithm variant's performance into a vector that reflects its performance on different problems. The second, Shapley-based meta-representations, was derived by training regression models to predict algorithm performance based on problem landscape features and calculating Shapley values [11] to determine the importance of each feature in these predictions.

Using performance-based meta-representations and assessing their distributions, we analyzed how different module configurations affect algorithm performance and identified which modules have the most significant impact. Using the Shapley-based meta-representations, we investigated which problem landscape features are most influential in predicting performance across different module configurations. Finally, we trained different feature-based classifiers to predict the module configurations of algorithm variants based on their meta-representations, aiding in the identification of algorithms with similar behaviors when configuration details are unknown.

1.8.2.3 Scoring-based knowledge graph embeddings for algorithm performance prediction

While the above-mentioned approaches focused on exploiting benchmarking data represented in standard tabular format, in this dissertation we also assessed the effectiveness of utilizing KGs derived from benchmarking data for predicting the performance of modular algorithms in BBO.

Our methodology begins with the construction of a KG that represents entities such as optimization problems, algorithm configurations, and their relationships. Each problem instance and algorithm variant is represented as a node in the KG, with edges capturing the semantic relationships between them, including performance outcomes. Performance links, labeled as "solved" or "not-solved", connect algorithms and problems based on whether the algorithm achieved the target precision within the given budget. Our goal is to determine whether these algorithms can solve specific optimization problems within predefined runtime budgets and solution quality thresholds.

We frame the performance prediction task as a knowledge graph completion problem, employing scoring-based knowledge graph embedding (KGE) techniques. Using the ComplEx model [12], we learn vector representations of entities and relations within the KG. During training, we use known performance links and apply negative sampling to generate negative examples, optimizing the model to assign higher scores to true triples.

In the inference phase, we predict missing performance links by calculating scores for both "solved" and "not-solved" relations between algorithm and problem nodes. The relation with the higher score is selected as the predicted outcome.

Our approach is evaluated across various scenarios, including different problem dimensions and performance thresholds. In balanced classification settings, the model demonstrates strong predictive performance, outperforming baseline methods. However, in imbalanced scenarios, the model's performance declines due to class disproportion. To address this, we extend our methodology by training a Random Forest classifier on top of the learned embeddings, which was shown to improve predictive performance.

1.8.2.4 Graph neural network for algorithm performance prediction

Scoring-based KGEs inherently operate in a transductive setup, which limits their ability to generalize to unseen data or nodes. To overcome these limitations, we explored the use

of Graph Neural Networks (GNNs) for algorithm performance prediction.

Using the same benchmarking data on modular optimization algorithms, we constructed a heterogeneous graph representing the relationships among optimization problems, algorithm configurations, parameters, and performance. Nodes in the graph represented entities such as problems and algorithms, while edges captured various types of relationships, creating a rich and complex relational structure.

We adapted message-passing GNN architectures to handle the graph's heterogeneity by implementing relation-specific convolutions for each edge type. Specifically, we experimented with two GNN models: GraphSAGE [13] and Graph Attention Networks (GAT) [14]. Our GNN models performed node regression, predicting continuous performance values for the performance nodes rather than treating the problem as a classification task (e.g., solved vs. not solved). The architectures included multiple GNN layers for message passing and relation-specific aggregation.

To enhance the interpretability of our models, we applied GNNExplainer [15], which helped identify influential graph structures and node features affecting the predictions.

Our methodology demonstrated that GNNs can effectively predict algorithm performance in an inductive setting, generalizing to unseen problems by leveraging both relational structures and node features.

1.9 Structure of the Dissertation

This introductory chapter provides an overview of the research domain explored in this dissertation and outlines the motivation behind this work. It also identifies the research gaps and limitations that this work aims to address. Furthermore, the chapter defines the research questions formulated to overcome these challenges and highlights the main scientific contributions. The rest of the dissertation is organized as follows.

In Chapter 2, we provide the background necessary for understanding the methods presented in this dissertation. The chapter begins by defining key concepts in the domain of knowledge representation and introducing state-of-the-art principles for data management. We then delve into the domain of machine learning (ML), covering essential concepts in ML benchmarking and introducing the domain of meta-learning. Subsequently, we explore the field of black-box optimization, emphasizing key benchmarking concepts and introducing modular optimization algorithm frameworks, which are examined in detail later in this dissertation. The chapter also defines the task of algorithm selection and explains its practical significance. Finally, we conclude the chapter with an overview of knowledge graph reasoning methods, focusing on scoring-based knowledge graph embedding (KGE) techniques and message-passing graph neural networks (GNNs).

In Chapter 3, we begin by formulating the problem addressed in this chapter and highlighting the importance of formal mechanisms for storing knowledge and data from the domain of MLC benchmarking. We also emphasize the need for semantic catalogs that facilitate the exploration of such data. Next, we review related work and outline existing MLC repositories. We then present the core contribution of this chapter: a semantic annotation schema for MLC benchmarking data. This schema introduces a comprehensive vocabulary for annotating MLC datasets, their associated metadata, MLC experiments, and performance data. Next, we demonstrate the practical application of the proposed schema through a proof-of-concept implementation in an online catalog for MLC benchmarking data, named MLCBench. Finally, we conclude the chapter with a summary of the contributions and a discussion of their implications.

In Chapter 4, we introduce the OPTION (OPTImization algorithm benchmarking ONtology) ontology, a semantic framework designed to address challenges in representing and

integrating BBO benchmarking data. The chapter begins by defining the problem and motivating the need for formalized, interoperable data management in this domain. We then review related work, highlighting gaps that OPTION aims to address. Next, we delve into the design and implementation of the ontology, describing its structure, key entities, and the semantic relationships that enable the representation of optimization algorithms, benchmark problems, problem landscapes, and performance data. To demonstrate its utility, we provide several practical use cases, including semantic annotation of data from prominent platforms like COCO and Nevergrad, as well as annotations for modular algorithm frameworks. Finally, we present the OPTION system for the annotation, storage, and querying of semantically enriched benchmarking data, including its integration within the IOHprofiler environment. The chapter concludes with a discussion on the ontology's extensibility and its potential to advance data sharing and interoperability in the optimization community.

In Chapter 5, we explore the task of algorithm selection for multi-label classification. We begin by defining the problem and emphasizing its relevance, followed by a review of related work. Next, we present various machine learning approaches for AS, including regression, classification, pairwise variants, single- versus multi-output models, and cost-sensitive models. The chapter continues with a description of the experimental setup, followed by a presentation of results and a discussion on the explainability of AAS models. We conclude with a summary of our findings.

In Chapter 6, we explore the use of modular optimization frameworks to assess the performance contributions of individual algorithm modules. The chapter begins by introducing the problem and its relevance, followed by a review of related work on modular frameworks, algorithm performance prediction, and explainable machine learning. Next, we detail the methodology, outlining the construction of performance-based and Shapley-based meta-representations for capturing algorithm behavior and linking it to problem landscapes. This is followed by the experimental design, which includes benchmarking algorithm variants across different problems, dimensionalities, and runtime budgets. The results section examines the impact of specific modules on performance, the role of problem landscape features on the prediction of algorithm performance, and the ability of classifiers to predict module configurations based on learned meta-representations. We conclude with a summary of our findings.

In Chapter 7, we explore the use of knowledge graphs (KGs) for predicting the performance of black-box optimization algorithms. We start by defining the problem and discussing the suitability of KGs for representing the symbolic relationships between optimization problems, algorithm configurations, and their performance. We continue by detailing the methodology, including the construction of KGs using the OPTION ontology and the implementation of a knowledge graph completion pipeline based on ComplEx embeddings. Next, we present the results of our experiments, highlighting the challenges of imbalanced classification and proposing a solution involving a Random Forest model trained on the learned embeddings. Finally, we conclude with a summary of our findings, a discussion of the limitations, and suggestions for future research to enhance KG-based performance prediction and extend its applicability to broader optimization contexts.

In Chapter 8, we investigate the use of Graph Neural Networks (GNNs) for predicting algorithm performance in numerical black-box optimization. We begin by framing the problem, contrasting the limitations of transductive approaches explored in Chapter 7 and emphasizing the advantages of GNNs for inductive tasks. Next, we describe the methodology, detailing the construction of a heterogeneous graph representation for the BBO benchmarking data, the GNN architecture, and the training process. We then outline the experimental setup. We next present the results, comparing GNN-based approaches

to traditional methods, such as Random Forest regressors, and discussing insights gained from explainability techniques like GNNExplainer. The chapter concludes by summarizing its findings and highlighting the potential of GNNs in this domain.

Finally, Chapter 9 provides the final conclusions of the work presented, a summary of the contributions, and a discussion of directions for future work.

Chapter 2

Background

In this chapter, we present the foundational concepts and background relevant to this dissertation. We begin by exploring the field of knowledge representation, emphasizing its role in AI and examining representational formalisms such as ontologies. Following this, we provide an overview of benchmarking practices in the domains of machine learning (ML) and black-box optimization (BBO), discussing key concepts and methodologies. We then delve into the meta-algorithmic task of algorithm selection, outlining its significance and approaches. Finally, we conclude with an overview of key concepts on learning from graph representations of data, highlighting the use of graph neural networks and knowledge graph reasoning.

2.1 Knowledge Representation

Knowledge Representation (KR) is a field in Artificial Intelligence (AI) that focuses on the design of formalisms that are computationally adequate for expressing knowledge about a particular domain [16]. The goal of KR is to create models and structures that represent knowledge in a way that it can be understood and processed automatically by computational systems. The effective representation of knowledge is crucial for tasks such as knowledge reasoning [17] and decision-making [18] in various real world systems.

In the field of AI, multiple representational frameworks have been employed for knowledge representation, including semantic networks, frames, rule-based systems, and ontologies. Among these, in recent times, ontologies have emerged as the predominant formalism. **Ontologies** can be defined as "explicit formal specifications of the concepts and relations among them that can exist in a given domain" [19]. The explicit semantic assumptions used in representing domain knowledge ensure a shared understanding of the domain, and the use of formal knowledge representation mechanisms makes it machine-processable.

Ontologies take an object-oriented view of modeling and include notions like: (1) *classes,* a set of semantically defined concepts; (2) *individuals,* instances of classes; and (3) *properties,* binary relations used to associate the classes and/or the individuals. It is important to distinguish between two types of ontology statements: TBox and ABox. *TBox* statements form the "terminology component" and describe the domain of interest by defining the classes and properties that form the vocabulary of the ontology (analogous to object-oriented classes). *ABox* statements are the "assertion component" facts associated with the TBox (analogous to instances of object-oriented classes). By assigning semantic meaning to knowledge facts and explicitly linking them to ontology terms, we perform the task of semantic annotation, thus effectively creating the ABox. Finally, a Knowledge Base (KB) integrates the TBox with the ABox, which allows for a comprehensive representation of domain knowledge, facilitating structured reasoning and querying.

All KBs that use the same ontology are interoperable, which means that distributed, heterogeneous systems and databases can easily interconnect and exchange information. Using the same ontology when annotating data and creating KBs facilitates automatic data integration, where classes and properties defined in the ontology serve as connecting points of the KBs.

Ontologies provide the basis for an unambiguous, formal representation of domain knowledge usually approved by experts in the domain. They provide the means for knowledge and data representations that are semantically understandable and available in machine-processable form. Thus, ontologies play a crucial role in sharing a common understanding of information structure among people or software agents. Apart from structuring domain knowledge in a principled way, ontologies improve data reusability by providing mechanisms for explicitly specifying provenance information of different resources [20]. Provenance information is the type of information that describes the origin of a resource, such as who created the resource, when it was published, and what license applies to its use.

Ontologies as computational artifacts are usually based on Description Logic (DL) as a knowledge representation formalism [16]. This logical component allows knowledge to be shared meaningfully at both machine and human levels. Also, an immediate consequence of having formal ontologies based on Description Logic is that they can be used in a variety of reasoning tasks and inference of new knowledge. For example, several reasoning engines can infer new knowledge from ontologies, such as Hermit [21] and Fact++ [22].

Numerous applications from different domains that involve big data handling are based on ontology as a data model. Such applications can be found in biomedicine [23], food and nutrition [24], environmental studies [25], etc. Ontologies are also used to represent computer science domains, such as the domains of data mining and machine learning [26].

Ontologies as informational artifacts have been used to develop machine-readable, semantically interoperable data, which is essentially the central goal of the Semantic Web [27]. The Semantic Web has been trying to achieve this goal via semantic annotation of data found on the web with terms defined in ontologies, intended to give data a well-defined meaning. As a result, many technologies have been developed, including RDF, RDFS, triplestores, SPARQL, etc. Here, we describe the Semantic Web technologies relevant to this dissertation.

Resource Description Framework (RDF)[1] is a standard data and metadata exchange format. Its fundamental structure, the RDF triple, provides a simple and intuitive way to express different statements in the form of subject-predicate-object (s, p, o) expressions. Each triple encodes a single piece of information about a resource being described (the subject) by ascribing to it a property (the predicate) and assigning it a specific value (the object, which may be either a resource or literal value). For instance, the statement "John knows Peter" is encapsulated within an RDF triple as (John, knows, Peter), with "John" serving as the subject, "knows" as the predicate, and "Peter" as the object. Within a collection of RDF triples, resources are uniquely identified and may appear as either subjects or objects across various triples, thereby constructing an RDF graph. An RDF graph G is composed of a collection of RDF triples, structured in the form (s, p, o). This graph is depicted as a directed labeled graph, with edges represented as $s \xrightarrow{p} o$, indicating the directional relationship from subjects to objects via predicates.

RDF Schema (RDFS)[2] is another semantic technology standard that is an extension of the RDF data model and provides essential elements for describing ontologies, such as

[1] Resource Description Framework: `https://www.w3.org/RDF/`

[2] W3C RDF Schema 1.1: `https://www.w3.org/TR/rdf-schema/`

classes and properties (relations). The Web Ontology Language (OWL)[3] is a collection of representation languages for authoring ontologies with different levels of expressivity.

RDF triples are stored in *triplestores*, a specialized type of NoSQL database that resembles graph databases, which use graph structures to represent data. Triplestores can handle vast amounts of RDF records, often reaching trillions, making them highly suitable for applications within the Semantic Web. Compared to relational databases, triplestores offer several key advantages: they provide flexibility, as data can be easily altered since there is no predefined data schema; they support a dynamic, extendable data model also due to the Open World Assumption, which states that a lack of knowledge does not imply falsity, allowing for new information to be included without assuming completeness; and they facilitate efficient querying, easy import/export of triples and data sharing.

There are many implementations of triplestores, including Apache Jena TDB[4], GraphDB[5], AnzoGraph [6], Stradog[7] and Virtuoso [28]. SPARQL Protocol and RDF Query Language[8] is the standard language for querying and manipulating data stored within RDF triplestores. As established by the World Wide Web Consortium (W3C), SPARQL allows for the formulation of complex queries that can retrieve and manipulate data stored in RDF format across various RDF databases, providing a powerful means for working with the web of data in accordance with the principles of the Semantic Web.

2.2 Data Management Guiding Principles

n addition to knowledge representation, effective data management and stewardship are critical for ensuring that resources can be efficiently utilized and maintained over the long term. Data management involves the strategic and operational practices required to oversee the data lifecycle, ensuring its quality, accessibility, security, and usability [29]. In contrast, data stewardship focuses on the responsibility and accountability for managing specific data assets [30].

The FAIR (Findable, Accessible, Interoperable, and Reusable) [6] and TRUST (Transparency, Responsibility, User Focus, Sustainability, and Technology) [31] principles provide comprehensive guidelines for the management of data and repositories/catalogues, ensuring that they meet the needs of users while adhering to standards for sustainability and interoperability.

The FAIR principles are a set of guidelines designed to improve the management and stewardship of digital assets, particularly data. They are intended to ensure that data is organized in such a way that it can be easily found, accessed, integrated, and reused by both humans and machines.

By adhering to the FAIR principles, data becomes:

- **Findable** through the use of persistent identifiers and metadata, enabling both humans and machines to locate relevant datasets efficiently.
- **Accessible** by ensuring well-defined and clear access protocols, whether the data is open or protected.
- **Interoperable** through the adoption of standardized formats and ontologies, allowing datasets from different systems and sources to integrate seamlessly.

[3]W3C OWL2: `https://www.w3.org/TR/owl2-overview/`
[4]Apache Jena TDB: `https://jena.apache.org/documentation/tdb/`
[5]GraphDB: `https://graphdb.ontotext.com/`
[6]AnzoGraph: `https://cambridgesemantics.com/anzograph/`
[7]Stradog: `https://www.stardog.com/`
[8]W3C SPARQL 1.1: `https://www.w3.org/TR/sparql11-query/`

- **Reusable** by providing detailed documentation, licensing information, and clear provenance, ensuring that future users can effectively apply the data in new contexts.

The TRUST principles complement the FAIR principles by ensuring that the data repository/catalogue is:

- **Transparent** about its processes, offering clear documentation on how data is curated and maintained.
- **Responsible**, with clearly defined roles for those who manage and oversee the repository, ensuring the data's integrity and longevity.
- **User-focused**, designed to meet the needs of its users by providing intuitive interfaces and comprehensive support for data exploration and analysis.
- **Sustainable**, both financially and technologically, ensuring long-term access to and preservation of data.
- **Technology-driven**, utilizing reliable, scalable, and interoperable technologies that facilitate data storage, access, and integration with other systems.

Together, these principles form a comprehensive framework for managing data repositories/catalogues in a way that promotes long-term usability, transparency, and user trust.

2.3 Benchmarking for Machine Learning

2.3.1 The machine learning domain

Machine learning is a branch of AI that involves the use of algorithms that enable machines to emulate human intelligence by learning from data and past experiences without being explicitly programmed to do so [32]. In recent years, the availability of vast computational power and large datasets has significantly boosted the popularity of ML across various fields, such as healthcare, finance, and technology.

In ML, generally, we distinguish between three main types of learning from data:

- Supervised learning uses labeled data, where each instance in the dataset is associated with a corresponding label. This label, often referred to as the output, dependent, or target variable, is what the model aims to predict. The independent variables, or input data, are used to train the model. The objective is to develop models that can accurately predict the target variable on new, unseen data. During inference, the independent variables are provided as input to the trained model, and the output is the model's prediction.
- Semi-supervised learning builds on supervised learning by incorporating both labeled and unlabeled data in the training process. While we still aim to predict a target variable, semi-supervised learning utilizes a larger portion of unlabeled data, which is often costly to label. This unlabeled data is leveraged to improve the model's accuracy by exploiting the underlying structure of the data, enhancing the predictive power beyond what is achievable with labeled data alone.
- Unsupervised learning deals with datasets that contain only input variables without corresponding output labels. The aim is to uncover hidden patterns or intrinsic structures within the data, such as clustering similar instances together or reducing dimensionality.

In (semi-)supervised learning, based on the type of target variable, ML tasks can be categorized into primitive or structured output tasks. Primitive output prediction tasks involve predicting a single target variable, such as in classification (predicting discrete values) or regression (predicting continuous values). Structured output prediction tasks, on the other hand, involve predicting multiple interrelated target variables. Examples of such tasks are multi-target regression [33], multi-label classification [34], and hierarchical multi-label classification [35], [36].

Typically, to train and evaluate a ML model, datasets consisting of data examples with features (input variables) and labels (output variables) are used. Features are the measurable properties or characteristics of the phenomenon being observed, while labels are the target values that the model aims to predict. In classical ML, features are manually engineered to optimize model performance. However, in deep learning [37], these features are learned automatically by the model during the training process.

Over the years, various ML algorithms have been developed, each with its strengths and weaknesses. Below, we provide an overview of several popular algorithms, although many more exist beyond this selection.

Decision trees are a type of ML algorithm used for both classification and regression tasks [38]. They work by splitting the data into subsets based on the value of input features, creating a tree-like model of decisions. Each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (classification) or a continuous value (regression). Decision trees are easy to understand and interpret and can handle both numerical and categorical data.

Tree ensembles combine multiple decision trees to enhance performance and robustness compared to individual trees. One popular tree ensemble method is random forests (RF) [39]. RF is a powerful ensemble learning method that combines multiple decision trees. It uses bootstrap aggregating, or "bagging" to create multiple subsets of the original dataset through random sampling with replacement. Each subset is then used to train a separate decision tree. To introduce further randomness, RF select a random subset of features at each split in the decision trees, choosing the best feature from this subset to make the split. This process reduces the correlation between the individual trees, making the ensemble model more robust and preventing overfitting. In the prediction phase, random forests aggregate the results of the individual trees by taking a majority vote for classification tasks or averaging the predictions for regression tasks.

Artificial neural networks (ANNs) are ML models inspired by the human brain, consisting of interconnected units called neurons organized into layers [40]. These layers include an input layer, one or more hidden layers, and an output layer. Each neuron in a layer receives input, processes it using an activation function, and passes the result to the next layer. The connections between neurons have associated weights, which are adjusted during training to minimize error and improve the network's performance. This adjustment process, known as optimization, typically involves algorithms like gradient descent to find the optimal set of weights that best map the input data to the desired output.

ANNs with many hidden layers are called deep neural networks (DNNs) [41]. DNNs have given rise to the "deep learning" subfield in ML, where "deep" refers to the large number of layers in these networks. DNNs are a driving force behind modern AI systems and have achieved remarkable success in various complex tasks due to their ability to learn hierarchical representations of data. They are powerful for handling complex patterns and large datasets, making them excellent for tasks such as image recognition and natural language processing. Additionally, DNNs perform automatic feature engineering, extracting relevant features from raw data without manual intervention. However, they often suffer from a lack of interpretability, making it challenging to understand how they arrive at spe-

cific decisions, which can be a significant drawback in applications requiring transparency and explainability.

2.3.2 Key concepts of ML benchmarking

In ML, benchmarking refers to the systematic evaluation and comparison of ML methods against established "benchmark" datasets that serve as standards in the field [42]. Benchmarking helps determine whether a newly developed method outperforms the current state-of-the-art methods on a given task. Additionally, benchmarking identifies the strengths and weaknesses of different ML methods, offering insights into their comparative performance and suitability for various tasks. The primary goal of benchmarking is to establish a baseline or reference point, enabling the assessment of how well different models perform relative to each other under the same conditions.

Benchmarking studies are conducted through computational experiments, which generate valuable experimental data. Benchmarking data includes all the information related to a benchmarking study, such as ML datasets, metadata describing the properties of these datasets, details about the ML methods and their parameters, and specifics of the evaluation scenario. Additionally, it encompasses data measuring how different algorithms perform across various metrics. This comprehensive collection of benchmarking data is essential for thoroughly understanding and comparing the capabilities of different ML methods under standardized conditions.

To further detail the ML benchmarking process, we explore two key components: benchmark problems and performance metrics.

2.3.2.1 ML benchmark problems

Standardized datasets, often referred to more generally as problems, are essential for benchmarking machine learning methods. These benchmark datasets can include well-studied real-world data from various problem domains or synthetic data that is artificially generated to contain known underlying patterns. The use of synthetic data allows for controlled and precise evaluations.

2.3.2.2 ML performance metrics

The choice of performance metrics is a crucial element in benchmarking, as it directly impacts the assessment of algorithm performance. Comparisons can be made across a range of predictive performance evaluation metrics. Additionally, factors such as computational complexity, model training time, inference speed, computational resource usage, and model interpretability are important considerations in benchmarking studies. This comprehensive approach to benchmarking is essential for demonstrating the capabilities of ML methods.

2.3.3 Meta-learning and meta-data

Meta-learning (MtL) is a subfield of machine learning (ML) that systematically analyzes the performance of various ML approaches across a diverse set of learning problems. The primary goal of meta-learning is to learn from these experiences to improve the learning process itself, or in short, to "learn how to learn" [4]. By leveraging these accumulated experiences, often referred to as meta-data, MtL facilitates quicker and more efficient learning on new problems. Essentially, MtL involves any type of learning that utilizes prior experience from other problems.

Benchmarking is critical in MtL, as it provides essential meta-data through the systematic evaluation and comparison of different ML methods.

MtL typically involves training a meta-learner on a variety of learning problems, where each problem is represented by a separate dataset with its own specific goal or learning objective. The meta-learner extracts patterns and knowledge from these problems, which can then be applied to optimize performance on new, unseen problems. This process can significantly reduce the computational resources and time required to develop high-performing models for new problems.

The challenge in MtL is to systematically learn from prior experiences in a data-driven manner. First, through a benchmarking process, we need to collect meta-data that describe previous learning problems. Second, we need to utilize this meta-data to extract and transfer knowledge, guiding the search for optimal models for new problems.

The meta-data includes information such as algorithm configurations and its hyper-parameters, model evaluations (e.g., accuracy and training time), learned model parameters (e.g., weights of an ANN), and measurable properties of the problems themselves, known as meta-features. Meta-features describe characteristics of the datasets and the associated learning problems, helping to understand their complexity and nature [43]. These meta-features are used as input features when training a meta-learner.

Meta-features can be broadly categorized into five categories [43]:

1. Simple: These measures are easily extracted from the data, commonly known, and do not require significant computational resources. They include basic statistics such as the number of instances and the number of features.

2. Statistical Features: These capture the statistical properties of the data, such as mean, variance, skewness, kurtosis, and correlations.

3. Information-Theoretic Features: These features are derived from information theory and are based on entropy. They measure the amount of information and complexity within the dataset.

4. Model-Based Features: These features are derived from the performance of simple models on the dataset, such as decision trees.

5. Landmarking Features: These features involve training simple and fast algorithms on the dataset and using their performance as indicators. The algorithms should have different biases to capture diverse aspects of the data, and they must operate with low computational cost.

Meta-learning has shown promising results in various tasks, including algorithm performance prediction [44], algorithm selection [45], and algorithm configuration [46].

2.4 Benchmarking for Numerical Black-Box Optimization

Optimization is the process of finding the best possible solution for a given problem by systematically adjusting input parameters to maximize or minimize a specific objective function [47]. It aims to achieve an optimal result according to defined criteria and constraints, often by balancing competing factors to find the most effective solution. Optimization methods are used across scientific and industrial fields to improve processes, design efficient systems, and make informed decisions based on quantitative goals.

2.4.1 The domain of black-box optimization

Black-box optimization problems are a specific class of optimization problems where the structure of the objective function is unknown, unexploitable, or non-existent [48]. In such cases, we do not have any prior knowledge about the underlying structure of the problem, and therefore, we treat it as a "black box".

In this dissertation, we focus on single-objective numerical or continuous optimization, meaning we deal with optimization problems where there is only one objective to optimize, and the objective function is real-valued. The term "single-objective" indicates that there is only one criterion or goal to optimize, distinguishing it from multi-objective optimization, where multiple criteria are considered simultaneously.

Given the nature of these problems, applying problem-specific algorithms is not feasible. Instead, they must be addressed using black-box optimization algorithms (BBOAs). One class of BBOAs is iterative optimization metaheuristics [49]. These algorithms are particularly well-suited for black-box optimization problems as they search for the optimal solution by iteratively querying the objective function with different inputs (i.e., *solution candidates*). They rely solely on this information to guide their search towards the most promising regions of the search space, without leveraging any knowledge of the underlying structure or characteristics of the problem [2].

Iterative optimization heuristics generally work as follows:

1. Initialization: The algorithm begins by selecting an initial set of candidate solutions, often referred to as the initial population.

2. Evaluation: Each candidate solution is evaluated using the objective function, and its quality is assessed.

3. Selection: Based on the quality assessments, a new set of candidate solutions is selected. This selection process often involves mechanisms inspired by natural processes, such as survival of the fittest in evolutionary algorithms.

4. Generation of New Solutions: New candidate solutions are generated from the selected ones through various operators like mutation, crossover, or other problem-specific methods.

5. Iteration: Steps 2 through 4 are repeated iteratively. With each iteration, the algorithm gathers more information about the problem, allowing it to focus on the most promising regions of the solution space.

6. Termination: The process continues until a termination criterion is met. This can occur when there has been no improvement in the population for a specified number of iterations, indicating stagnation. Alternatively, the process may terminate upon reaching an absolute number of generations of candidate solutions, thereby exhausting the function evaluation budget. The algorithm can also terminate when the objective function value achieves a pre-defined target, signifying that an optimal or satisfactory solution has been found.

Some commonly used iterative optimization heuristics include genetic algorithms [50], particle swarm optimization [51], and ant colony optimization [52], among many others. These algorithms are inspired by natural processes such as evolution and swarm intelligence, and they fall under the broader umbrella of Evolutionary Computation (EC) [53].

2.4.2 Key concepts of BBO benchmarking

Benchmarking in BBO, much like in the domain of ML, is crucial for evaluating and comparing the performance of different algorithms under standardized conditions [5]. Key components of BBO benchmarking include benchmark problems, performance metrics, and problem landscape analysis, each playing a distinct role in creating comprehensive assessments of algorithm capabilities.

2.4.2.1 BBO benchmark problems

To reliably estimate the performance of various optimization algorithms, they need to be tested on a diverse set of problem instances across different problem classes.

To facilitate the benchmarking of BBO algorithms, researchers have developed comprehensive test suites of benchmark problems. These suites include a variety of functions designed to assess different aspects of optimization algorithms.

Notable benchmark suites include the BBOB (Black-Box Optimization Benchmarking) suite [54], the CEC (Congress on Evolutionary Computation) competitions , MA-BBOB (Many-Affine Combinations of BBOB Functions) [55], and Nevergrad's suite [56] of benchmark functions, among others.

In this dissertation, we utilize the BBOB test suite available on the COCO (COmparing Continuous Optimizers) platform [57]. This suite includes 24 noiseless, single-objective test functions that are to be minimized. The functions can be tested across various dimensionalities, representing the number of variables in the problem, specifically $D \in \{2, 3, 5, 10, 20, 40\}$, and are defined within a search space ranging from $[-5, 5]^D$. The 24 functions are grouped into five distinct categories based on the properties they exhibit.

The five categories of the BBOB functions are as follows:

1. Separable Functions (F1–F5): These functions allow the optimization of variables independently, making them simpler and less computationally intensive.

2. Functions with Low Conditioning (F6–F9): These functions have a well-behaved landscape with moderate difficulty, suitable for evaluating algorithms' performance in less complex scenarios.

3. Unimodal Functions with High Conditioning (F10–F14): These functions have a single global optimum but are challenging due to their steep slopes and narrow valleys.

4. Multimodal Functions with Adequate Global Structure (F15–F19): These functions feature multiple local optima with a clear global structure, testing the algorithm's ability to escape local optima and find the global solution.

5. Multimodal Functions with Weak Global Structure (F20–F24): These functions are highly challenging due to their many local optima and weak or deceptive global structure.

To generate multiple instances of each of the 24 problem classes, various transformations are applied to the functions. These transformations include scaling, which involves multiplying the input variables by a scaling factor, and translation, which shifts the function's position within the search space by adding a constant vector to the input variables. The transformations ensure that the optimal solution is not always located at the same point. By applying these transformations, each function can exhibit a wide range of characteristics, making the benchmark suite more robust and comprehensive.

2.4.2.2 BBO performance metrics

In BBO, we lack an explicit formulation of the objective function. Therefore, we must evaluate the function using candidate solutions, often making this the most computationally expensive aspect of the optimization process. This is particularly true when function evaluations are time-consuming or resource-intensive, such as in simulations of complex physical systems or expensive experimental setups. Therefore, it is beneficial to use the number of function evaluations as a measure of BBOAs running time.

For measuring the performance of an optimization algorithm, a key metric is the target precision, which measures how close the algorithm's solution is to the known optimal solution of the benchmark problem. This is often expressed as the difference between the objective function value of the best-found solution and the objective function value of the optimum. In real-world problems, where the true optimum is often unknown, performance might be measured relative to the best-known solution or an estimated optimum based on expert knowledge or extensive computational effort.

There are two common approaches in evaluation of BBOAs: *fixed-target* and *fixed-budget*. In fixed-target evaluation, the goal is to reach a predefined target value of the objective function. The performance of the algorithm is measured by the number of function evaluations required to reach this target, providing insights into the algorithm's efficiency in approaching near-optimal solutions.

In fixed-budget evaluation, the algorithm is allowed to run for a predetermined number of function evaluations, which constitutes the budget. The performance is then measured by the best objective function value obtained within this budget. Fixed-budget evaluation is particularly relevant in real-world scenarios where computational resources are limited, and it is essential to understand the quality of solutions that can be obtained within those constraints.

2.4.2.3 Problem landscape analysis

To effectively characterize and utilize benchmarking problem instances in ML pipelines, detailed representations of these problems are essential. One such representation method is Exploratory Landscape Analysis (ELA) [58], which employs mathematical and statistical methods to derive features that describe the problem landscape. ELA is specifically designed to support the design of black-box optimization algorithms by providing a comprehensive set of features. These features can serve as inputs to ML models, which can then be used, for example, to recommend algorithms that best fit the problem at hand.

Since we are dealing with black-box optimization, these features must be derived from a set of samples from the problem instance. ELA features are calculated from a sample of points in the problem's search space. The calculation process involves several steps. First, a set of points is sampled from the search space of the problem instance. The sample size and sampling technique significantly impact the quality of the ELA features [59]. The sample size needs to be sufficiently large to capture the landscape's characteristics but small enough to keep computational costs manageable. Typically, random sampling or Latin Hypercube Sampling (LHS) [60] is used to ensure a diverse and representative sample of the search space.

Once the sampling is completed, the sampled points are evaluated using the objective function, and various statistical and mathematical methods are applied to compute the ELA features. These features include basic metrics such as the mean, variance, skewness, and kurtosis of the objective function values at the sampled points. Additionally, dispersion measures describe the spread and distribution of the objective function values, while level set features provide insights into the topology of level sets within the landscape by

measuring the proportion of points below a certain objective function threshold. Meta-model features are derived from fitting a simple model, like a linear or quadratic model, to the sampled data and analyzing the model's properties. Local search features are obtained from performing local searches starting from the sampled points and analyzing the results.

The R-package *flacco* [61] facilitates the computation of 343 Exploratory Landscape Analysis (ELA) features, grouped into 17 feature sets [45].More recently, a Python version, known as the *pflacco* library [62], has also been developed. In this study, we utilize a dataset [63] containing pre-calculated ELA features for various COCO benchmark problems, generated using the flacco library. These features have been applied and have demonstrated promising results in several automated tasks, such as predicting algorithm performance [64], [65], selecting appropriate algorithms [45], [66], and configuring algorithms automatically [67].

While ELA features are the most commonly used for analyzing problem instance landscapes, other approaches also exist [68]. Examples include topological landscape analysis [69], which characterizes problems using topological data analysis techniques [70], and deep learning-based approaches, which extract low-level features using various types of deep neural networks. These methods include features learned through convolutional neural networks [71], [72], transformer-based architectures like TransOpt [73] and DeepELA [74], and autoencoders such as DoE2Vec [75].

2.4.3 Modular algorithm frameworks

Benchmarking in black-box optimization provides a structured way to evaluate algorithms, but comparing variations within algorithms presents its own challenges. In evolutionary computation, new algorithms are often developed iteratively. Instead of introducing novel algorithmic ideas in isolation, they are typically presented as extensions to preexisting algorithms. Two prominent families of evolutionary optimization algorithms, Differential Evolution (DE)[76] and Covariance Matrix Adaptation Evolution Strategies (CMA-ES)[77], have been developed in an iterative manner. Since these two algorithms have been well-researched for over a decade, many variations and modifications have been proposed. Some of these modifications may be relatively minor, such as proposing an alternative initialization of the population. Larger changes may affect the structure of the algorithm by introducing restart mechanisms or new adaptation schemes for internal parameters.

Since most of these changes are proposed in isolation, it is often difficult to understand how these variations interact. All of this has led to the development of modular algorithms frameworks. In this dissertation, we focus on studying two such frameworks, modCMA-ES [78] and modDE [79], specifically designed for the CMA-ES and DE algorithms, respectively, although other frameworks also exist [80]–[83]. These frameworks combine large sets of variations into a single code base, where arbitrary combinations of variations can be combined into a variety of possible algorithm configurations. This not only allows a fair comparison between two different variations of the algorithm but also a more robust analysis of the potential interplay between algorithm components.

2.5 Algorithm Selection

Algorithm selection (AS), often referred to as per-instance algorithm selection, is a meta-algorithmic approach that selects the optimal algorithm (in terms of performance or runtime) from a portfolio of complementary algorithms for a specific problem instance [84]. AS is a general approach that can be applied across a variety of computer science domains, including optimization and machine learning, to improve performance by tailoring

the choice of algorithm to the problem's characteristics.

Let $\mathcal{A}$ be the set of all available algorithms in the portfolio, $\mathcal{I}$ the set of all problem instances and $c(a, i)$ the cost associated with running algorithm a on instance i. The goal is to learn a mapping $s : \mathcal{I} \to \mathcal{A}$ that assigns an algorithm to each instance so that the total cost $\sum_{i \in \mathcal{I}} c(s(i), i)$ summed over all instances is optimized (minimized or maximized). Some problems may require minimization (e.g., minimizing the runtime or error), while others might need maximization (e.g., maximizing accuracy or efficiency) of the cost function.

AS has been investigated across multiple domains, including BBO [85], [86] and automated machine learning (AutoML) [87]. Specifically within AutoML, AS is often considered as a meta-learning task [4]. In this context, the portfolio comprises a set of ML algorithms and the problem instances are datasets. The cost is typically a metric that measures the efficiency or effectiveness of the algorithm on that dataset. For instance, in a classification scenario, the cost metric might be accuracy, which we aim to maximize as a higher value indicates better performance. Alternatively, in a regression context, the focus might be on minimizing the error rate. These are just examples; the cost metric can vary depending on the specific requirements and objectives of the problem at hand. Consequently, the primary objective of AS is to accurately predict which machine learning algorithm will yield the best performance for each dataset, based on the chosen metric.

In BBO, the AS problem is approached similarly to ML, but instead of datasets, we typically have optimization problems where the objective function is not explicitly defined. Typically, the concepts and methodologies applied to AS in ML are equally applicable to AS in BBO.

There are different strategies for building an algorithm selector (e.g., parallel algorithm portfolios, algorithm schedules, or ML-based algorithm selection). The integration of ML techniques with AS has led to the development of Automated Algorithm Selection (AAS) [7]. In AAS, information about the problem instance is used as input, and ML techniques are applied to predict which algorithm will perform best on that specific instance. This method reduces the need for manual intervention and extensive human expertise. Problem instances in AAS are usually characterized by numerical features, which can range from basic statistics such as the number of variables or data examples to more sophisticated landmark features. These landmark features might involve training a simple ML model to extract deeper insights from the data, enhancing the predictive power of the algorithm selection process.

There are various machine learning approaches to solving the AAS problem. For example, the problem can be addressed through multi-class classification [88], where each class represents a different algorithm, and the model predicts the most suitable algorithm for a given instance. Alternatively, regression models can be used where the output predicts a performance metric of the algorithms [89]. The decision on which algorithm to select is then made by comparing these predicted metrics and choosing the algorithm that scores best according to the desired performance criteria.

Other machine learning approaches to AAS include clustering [90] and pairwise methods [91], [92]. Clustering groups similar problem instances together and identifies algorithms that consistently perform well within these groups. Pairwise approaches, whether classification or regression, involve comparing two algorithms at a time to determine which is more likely to yield better performance on a specific instance. The effectiveness of these approaches can vary significantly across different domains, and determining the most suitable method for a particular setting remains a topic of active research.

Most of the discussed approaches often rely on feature-based or tabular data and depend heavily on problem representation features. Recently, feature-free approaches utilizing deep

neural networks (DNNs) have also begun to emerge [93]. However, an intriguing direction in this context is the use of graph representations of all data for algorithm selection. By representing problem instances, algorithms, and their interactions as graphs, and employing graph representation techniques , it is possible to capture complex dependencies and relationships.

2.6 Knowledge Graphs and Knowledge Graph Reasoning

A Knowledge Graph (KG) is a structured representation of knowledge consisting of entities and the relationships between them. It is typically visualized as a semantic network or a directed graph structure, where nodes represent entities and edges represent the connections or relationships between these entities in a symbolic form. KGs store knowledge in units of facts, often in the form of triples (subject-predicate-object), where the subject and object are the nodes and the predicate is the edge. Ontology knowledge bases (KBs) can also be considered KGs due to their graph-based structure.

KGs play an important role in a variety of downstream applications, such as semantic search, recommendation systems, and question answering [94]. However, most of the real-world KGs are often incomplete and fail to include all relevant facts. Therefore, KG reasoning (KGR), also known as KG completion (KGC) is essential for the application of KGs. KGC improves the completeness of KG by inferring new knowledge and insights from existing knowledge graphs. For example, KGC can predict a missing relationship between two entities or identify missing head (subject) or tail (object) entities in a triple.

Knowledge graph embedding (KGE), also known as knowledge graph representation learning (KGRL), is a prominent research direction within Knowledge Graph Completion (KGC) that has rapidly gained significant attention [95]. KGEs are models that transform entities and relationships into low-dimensional vector representations that are designed to preserve the inherent relationships and structures within the knowledge graph. By enabling various downstream tasks such as link prediction, entity resolution, and clustering, KGEs significantly enhance the utility and performance of knowledge graphs in practical applications, such as news recommendation [96], e-commerce [97], medical recommendations [98], and remote sensing imagery classification [99]..

The following subsections provide an overview of two central approaches to KGR: scoring-based KGE methods, which use scoring functions to evaluate the plausibility of relationships, and graph neural networks (GNNs), which leverage the graph structure of KGs to capture complex patterns and interactions.

2.6.1 Scoring-based KGE methods

One popular approach to generating KGEs is through the use of scoring-based models. These models assign a score to triples based on their correctness. The goal is to generate KGEs that produce high scores for likely correct triples and low scores for less likely ones. During training, both positive and negative triples are used. Positive triples are the actual triples present in the KG, while negative sampling generates invalid triples by corrupting a valid triple, either by replacing the head or the tail entity with a random entity from the graph.

The embeddings are learned by minimizing a loss function that penalizes incorrect prediction. The optimization is performed using gradient-based methods like stochastic gradient descent or its variants. By iteratively updating the embeddings, the model learns to assign higher scores to valid triples and lower scores to invalid ones, thereby effectively capturing the underlying relationships in the knowledge graph.

Over the years, various scoring functions have been proposed, each with its own strengths and weaknesses. For example, many scoring models can only model certain types of relationships, such as symmetry, antisymmetry, composition, one-to-many, and many-to-one relationships.

In this dissertation 7, we will be applying the ComplEx scoring function [12] for training KGEs. ComplEx is an extension of the DistMult model [100], designed to address some of its limitations.

DistMult is a knowledge graph embedding model that represents entities and relationships as vectors in a high-dimensional space. It scores the likelihood of a relationship between entities by computing a weighted dot product, effectively capturing symmetrical relationships but struggling with asymmetrical ones due to its symmetric nature. ComplEx extends DistMult by using complex-valued embeddings, allowing it to handle both symmetrical and asymmetrical relationships. This is achieved by leveraging the properties of complex numbers, which enhance the model's expressiveness and ability to capture a broader range of relational patterns in the knowledge graph.

2.6.2 Graph neural networks

Graph Neural Networks (GNNs [101] are a class of neural networks specifically designed to operate on graph-structured data. Unlike traditional neural networks that work on regular grids like images (2D grids) or sequences (1D grids), GNNs are tailored to capture the underlying relationships and patterns in data that is represented as a graph, where entities (or nodes) are connected by relationships (or edges). This capability makes GNNs highly effective for tasks such as node classification, link prediction, and graph classification, particularly in domains where the data is inherently relational.

In a typical GNN, the input is a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges connecting pairs of nodes. Each node $v \in V$ may have associated features, often represented as a feature vector. GNNs operate by iteratively updating the node representations by aggregating information from neighboring nodes, allowing each node to capture local and global information from the graph structure.

The core of a GNN lies in its *message-passing mechanism*, where each node gathers and aggregates messages from its neighbors. This process is repeated across several layers, allowing information to flow and propagate throughout the graph, enabling nodes to capture and learn complex relationships. Each GNN layer typically operates in two key steps:

1. **Message:** Each node computes a message to send to its neighbors. The message is derived from the node's current feature representation, which comes from the previous layer (or the input features in the first layer). Formally:

$$m_u^{(l)} = \text{MESSAGE}^{(l)}\left(h_u^{(l-1)}\right), \quad u \in \mathcal{N}(v)$$

   Here, u represents a neighboring node of v, and $\mathcal{N}(v)$ is the set of neighbors of node v. The term $h_u^{(l-1)}$ refers to the hidden feature vector of node u from the previous layer (layer $l-1$), while l represents the current layer. The message $m_u^{(l)}$ is the result of applying a message function (such as a linear transformation) to the feature vector $h_u^{(l-1)}$.

2. **Aggregation:** Once a node receives messages from its neighbors, it aggregates them to update its own hidden representation. This involves applying an aggregation

function to combine the messages from the neighbors as well as the information from itself :

$$h_v^{(l)} = \text{AGGREGATE}^{(l)}\left(\{m_u^{(l)} : u \in \mathcal{N}(v)\}, m_v^{(l)}\right)$$

Nonlinearity (e.g., Rectified Linear Unit (ReLU), Sigmoid) is often applied during the **Message** and/or **Aggregation** step to increase the expressiveness of the model, enabling it to capture more complex patterns in the data.

At each layer, the hidden representation of a node is updated by combining its own features with those of its neighbors. This iterative message-passing process allows the GNN to learn richer and more complex relationships as the number of layers increases, enabling nodes to capture dependencies spanning multiple hops across the graph..

Many variants of GNNs have been proposed, each using different aggregation and update mechanisms to capture different graph structures and relationships. For example:

- **Graph Convolutional Networks (GCNs)**: The original GCNs, introduced by Kipf et al. [102], extended neural network convolution operations to graph-structured data, allowing information to propagate between neighboring nodes through graph convolution layers. This approach generates node embeddings based on the features of each node's neighborhood. Initially, GCNs were designed for transductive tasks, requiring access to the entire graph during training and limiting their ability to generalize to unseen nodes.

- **GraphSAGE**: GraphSAGE [13] extends the GCN framework by introducing a more scalable approach to learning node representations. Instead of aggregating information from all neighbors, as in traditional GCNs, GraphSAGE samples a fixed-size set of neighbors. This sampling method allows for efficient training on large graphs. The key idea behind GraphSAGE is learning how to aggregate feature information from a node's local neighborhood. By doing so, the model can generalize to unseen nodes, as it learns transferable patterns from the sampled neighborhoods rather than relying on the full graph structure.

- **Graph Attention Networks (GATs)**: GATs [14] incorporate the attention mechanism into GNNs, enabling nodes to assign different importance scores to their neighbors. This selective focus enhances the model's ability to capture relevant information during message passing, which is particularly beneficial in heterogeneous graphs where node and edge types can differ widely. Unlike GCNs, which treat all neighbors equally, GATs utilize self-attention layers to compute weights for each neighboring node. These weights are based on the similarity between the feature vectors of the source node and its neighbors, allowing the network to prioritize more informative connections.

The key difference between scoring-based KGE models like DistMult and ComplEx and GNNs lies in how they learn the embeddings. Scoring-based KGE models, such as DistMult, focus on learning embeddings for entities and relations by optimizing a scoring function that predicts the likelihood of a triple (head, relation, tail) being true. In contrast, GNNs like GraphSAGE and GAT do not train a separate embedding for each node. Instead, they learn a function that generates node embeddings by aggregating and transforming information from neighboring nodes and edges. This approach allows the model to dynamically generate embeddings based on the local graph structure, enabling better generalization to unseen nodes. GNNs are more flexible in handling various graph types and incorporate both structural and feature information from the graph.

Chapter 3

Semantic Catalogue of MLC Benchmarking Data

In this chapter, we detail the development of an ontology-based schema specifically designed for the semantic annotation of multi-label classification (MLC) benchmarking data. This schema is instrumental in constructing a semantic catalogue dedicated to organizing and facilitating access to MLC benchmarking data.

The chapter starts by defining the specific challenges that necessitate the creation of an ontology-based catalogue for MLC benchmarking data. This section outlines the problems associated with managing complex MLC data and the benefits an organized catalogue brings in terms of data discoverability, accessibility, interoperability, and reusability in accordance with FAIR principles [6]. Following the problem definition, we detail the MLC semantic annotation schemas we have designed. We then describe the development of a system for automatic data annotation, storage, and querying. Finally, we present the creation of an online version of the catalogue to enhance accessibility to MLC benchmarking data.

This chapter is based on the article "A Catalogue with Semantic Annotations Makes Multilabel Datasets FAIR" [103], published in Scientific Reports by Nature Publishing Group. The chapter builds upon the work presented in the paper by expanding the catalogue to include not only datasets but also other benchmarking data, such as experimental results and trained models.

All data, code, and resources developed for this chapter are publicly available on GitHub at: `https://github.com/KostovskaAna/MLC-Schema`.

3.1 Problem Definition

In the era of Big Data, the ability to efficiently manage, search, and analyze vast amounts of data becomes paramount. This holds particularly true for the field of MLC, which has seen growing interest due to its wide applicability in real-world problems that require the ability to assign multiple labels to a single instance. This capability, a defining feature of MLC, is crucial in domains such as text categorization, image and video annotation, genomics and bioinformatics, and semantic scene classification. The popularity of MLC is also attributed to the advancements of deep learning and other advanced ML techniques that have significantly improved the performance of MLC algorithms. As a result, a large volume of benchmarking data is being produced, facilitating the comparison of MLC algorithms' performance. Properly managing and providing easy access to this wealth of benchmarking data, including datasets, algorithms, and computational experiments, among others, is crucial for the progress of research and development of practical applications in MLC.

Data catalogues have emerged as fundamental components for the effective management of large volumes of data, serving as organized inventories of data assets. They provide metadata, which includes descriptions about the data stored in data lakes or other data storage solutions. However, creating data catalogues does not immediately solve all data management challenges. Specifically, issues of data findability, accessibility, interoperability, and re-use – the four principles of FAIR data [6] – remain unresolved.

To facilitate the principles of FAIR data, it is crucial to enhance data catalogues with a semantic layer. The semantic layer, usually in the form of an ontology, provides a rich, meaningful description of the data, covering aspects such as content of the data, data provenance, and access rights. These semantic descriptors, referred to as semantic annotations in the text and often structured as RDF graphs, support both FAIR and TRUST principles, two key benchmarks for effective data management and stewardship.

Semantic annotations allow for the contextual understanding of data. This means that data is linked based on its meaning and relationships, rather than just its format or source. Such an approach significantly enhances the findability and accessibility of data, enabling researchers to discover and integrate relevant data. Interoperability and reusability are enhanced through standardized formats and shared ontologies, promoting data exchange and reuse. Moreover, ontology-based catalogues embody the TRUST principles by fostering transparency in data management, ensuring responsible data stewardship.

Despite the growing significance of MLC in various domains, we still lack semantic catalogues of MLC benchmarking data. While there exist several repositories and catalogues of MLC datasets [104]–[108], these resources fall short in several key areas. They provide only a limited selection of benchmark MLC datasets and lack the capability to store experimental data for the comparison of MLC algorithms. Furthermore, these sources do not offer performance metrics specifically tailored for MLC, nor do they provide detailed information on dataset features or support the comparison of MLC landscape properties. Additionally, none of these resources incorporate a semantic layer to adhere to the FAIR data principles.

This gap highlights the need for dedicated semantic resources that can support the specific demands of MLC research and application. Creating an ontology-based catalogue for MLC benchmarking data is an initiative aimed at addressing this need. This resource would serve as a comprehensive hub for MLC datasets, experiments, and algorithms, facilitating in-depth analysis and comparison of MLC algorithms.

Objective: In this dissertation, our objective is to develop MLCBench, a semantic catalogue of MLC benchmarking data. We achieve this by designing an ontology-based semantic schema specifically tailored for MLC benchmarking data. Furthermore, we construct a knowledge base that forms the semantic layer of our MLCBench catalogue. We then develop a prototype web application that integrates the semantic schema and knowledge base. This tool is crafted to enable semantic data search and retrieval, along with interactive exploratory data analysis, streamlining the exploration and utilization of MLC benchmarking data. The MLC semantic annotation schema developed in this work is designed to address the competency questions outlined in Table 3.1. The creation of the MLCBench catalogue is guided by a set of specific requirements, as detailed in Table 3.2. These requirements form the foundation for the catalogue's design and functionality, ensuring it meets the needs of the MLC research community.

3.2 Related Work

Over the years, verious MLC data repositories and catalogues have been developed. The Cometa repository [105], for instance, comprises 74 MLC datasets, characterizing them

Table 3.1: List of competency questions guiding the development of the ontology-based semantic annotation schema for MLC benchmarking data.

N^0	Competency question
1	What is the provenance data available for a specific MLC dataset?
2	What are the values of the MLC meta-features that depict the landscape of the MLC datasets?
3	Which datasets and algorithm were used in a specific experiment, and what was the experiment workflow?
4	How does a predictive model perform according to a chosen metric?
5	Which sampling technique was employed to create different splits of the dataset?
6	What hyperparameters were used for training the predictive model in a given experiment, and which algorithm was used?

with 12 meta-features. It also provides list of the descriptive features, target labels, and citations to their original publication. The KDIS repository [106] offers a slightly larger collection with 78 datasets, providing dataset domain categorization, 9 meta-features, and textual dataset descriptions. It stands out for the variety of data splits it provides such as random train-test, stratified train-test, random 5-fold CV, stratified 5-fold CV, stratified 10-fold CV. With a collection of 26 MLC datasets, Mulan [107] provides dataset domain categorization and 7 meta-features for characterizing the datasets. alongside download links to the data. However, these repositories are solely focused on datasets and do not include links to experiments conducted with these datasets or the resulting performance benchmark data.

Table 3.2: List of requirements guiding the development of the semantic catalogue and system for MLC benchmarking data.

N^0	Requirement
1	The system should support the semantic annotation of MLC benchmarking data, including datasets, algorithms, experiments, and experiment results, using the defined annotation schema.
2	The system should integrate and process data from multiple sources, ensuring interoperability across various MLC benchmarking datasets and related metadata.
3	Semantic annotations should be stored in a dedicated semantic data store to enable efficient querying and management.
4	The catalogue should provide an intuitive user interface that enables easy search, retrieval, and analysis of MLC benchmarking data.
5	The catalogue should offer advanced search capabilities that leverage semantic annotations for precise data retrieval.
6	The system should include interactive data visualization tools to facilitate dynamic exploration and cross-comparison of meta-features that describe the landscape of MLC datasets.
7	The system should provide visualization tools for exploring experiment outcomes across various MLC algorithms and performance metrics.
8	The system should be designed to be scalable, accommodating the increasing volume of MLC benchmarking data over time.

The Extreme Classification repository [108] stands apart by offering benchmark results

for 6 performance metrics, however, these are only accessible within GitHub repositories, which hinders data accessibility. The Extreme Classification repository describes the datasets with 6 meta-features and facilitates data access through download links. Nonetheless, all four repositories lack a semantic layer, which is crucial for enabling semantic searches, data understanding, and automated integration, thus presenting limitations in adherence to the FAIR principles of data management.

In the broader field of ML, there are several platforms have been developed to facilitate the sharing of ML datasets [42], [109], [110]. Among these, the OpenML platform [110] distinguishes itself as a comprehensive open-source project that enables researchers to share not only datasets, but also algorithms, and experiments. Despite its robust framework, OpenML's primary limitation is its broad focus. While it comprehensively supports tasks like regression, binary classification, and multi-class classification, it lacks features and tools tailored to MLC.

Although semantic catalogues have yet to be adopted in MLC and ML, their success in other fields highlights their potential for improved data management. Research indicates that incorporating semantic layers into data catalogues significantly enhances data discovery, integration, and use [111].

One notable implementation of semantic technologies is PubChem, an open-access chemistry database managed by the National Institutes of Health (NIH) [112]. PubChem has established itself as an indispensable resource for scientists, educators, and the public, serving millions of users globally each month. The database's adoption of PubChemRDF and SPARQL endpoints for querying demonstrates the practical application and benefits of semantic web technologies in facilitating accessible, machine-readable data. Similarly, Robert Bosch GmbH's introduction of the DCPAC Ontology within their data lake architecture exemplifies corporate adoption of semantic technologies [113]. Further illustrating the versatility of semantic technologies, the application of semantic technologies in a data lake designed for managing datasets from sensors or simulation programs in the manufacturing sector highlights the adaptability of these technologies [114]. These examples represent just a snapshot of the numerous platforms utilizing semantic data management technologies. Their effective application across various domains emphasizes the potential advantages of developing semantic catalogues in the MLC domain.

3.3 Semantic Annotation Schemes for MLC Benchmarking Data

In this section, we present semantic annotation schemes designed for MLC benchmarking data. Our focus is on the comprehensive annotation of MLC datasets and MLC experiments, which includes the performance data generated from these experiments. The MLC schema is publicly accessible at its persistent URL: `http://purl.archive.org/mlc-schema`.

3.3.1 Semantic annotation of MLC datasets

For semantic annotation of datasets, we have designed an ontology-based schema that enables the description of multiple aspects of MLC datasets. The schema is an adaptation of a more general annotation schema that covers a broader range of machine learning tasks presented in [115]. We can broadly categorize the semantic annotations into two groups: (1) Annotations of datasets with provenance information and (2) Annotations that capture relevant machine learning characteristics of the datasets.

Table 3.3: The list of Schema.org properties used for semantic annotation of MLC datasets with provenance details.

Property	Expected Type	Description
name	Text	A descriptive name of the dataset.
description	Text	A short summary of the dataset.
sameAs	URL	URL of a reference Web page that provides additional information about the dataset, which is usually stored in a different repository.
identifier	Property Value/ Text/URL	Identifier of the dataset such as Digital Object Identifier (DOI) or Compact Identifier. A dataset can have more than one identifier.
keywords	Text	Keywords or tags used to describe the dataset.
creator	Organization/ Person/Text	The creator of the dataset.
license	CreativeWork/ URL	Identifies a license document under which the dataset is distributed.
variableMeasured	Text	The variableMeasured property (referred to as unit of measurement later in the text) can indicate the variables that are measured in some dataset.
genre	Text	Genre (referred to as dataset domain later in the text) of the creative work (i.e., the dataset).
distribution	DataDownload	A downloadable form of this dataset, at a specific location, in a specific format. This property can be repeated if different variations are available.

Provenance information refers to the kind of information that describes the origin of a resource, which in our context is MLC dataset. It encompasses details such as the creator of the resource, when was it published, and what is its usage license, to name a few. For semantic description of provenance information, we have chosen the Schema.org vocabulary[1], a collection of schemas widely used for providing structured data on the Web. Schema.org is a collaborative, community activity with a mission to create, maintain, and promote schemas for structured data on the Web. Its widespread acceptance and implementation across various platforms make it a reliable choice for enhancing the discoverability and interoperability of web-based resources, such as our web-based MLC benchmarking data catalogue. Specifically, for the annotation of MLC datasets, we employ the `Dataset` schema [116] from Schema.org [117]. This schema is particularly suitable for our purposes as it is designed to structure data about datasets, making information like the dataset's name, description, identifier, and license readily accessible. The full list of properties we use for annotation of MLC datasets is presented in Table 3.3.

From an ML perspective, various types of annotations are considered relevant, including dataset specification, learning task, datatypes, and meta-features. To this end, to enable annotation of ML-specific information, we have integrated ontological concepts from two external ontologies, i.e., the ontology of core data mining entities (OntoDM-core) [26], and the generic ontology of datatypes (OntoDT) [118]. This integration has led to the development of a semantic annotation schema specifically designed for MLC datasets. Additionally, we have extended the OntoDM-core ontology and added concepts relevant to our domain, such as concepts that semantically define the MLC meta-features.

Our proposed schema is designed for tabular MLC data, often referred to as feature-based data. While MLC tasks are also prevalent in other areas, such as computer vision – where the goal is to label images with various objects – our current focus is on feature-based data. This schema is adaptable, and can be easily expanded support applications in computer vision and similar fields.

[1]Schema.org vocabulary: https://schema.org/

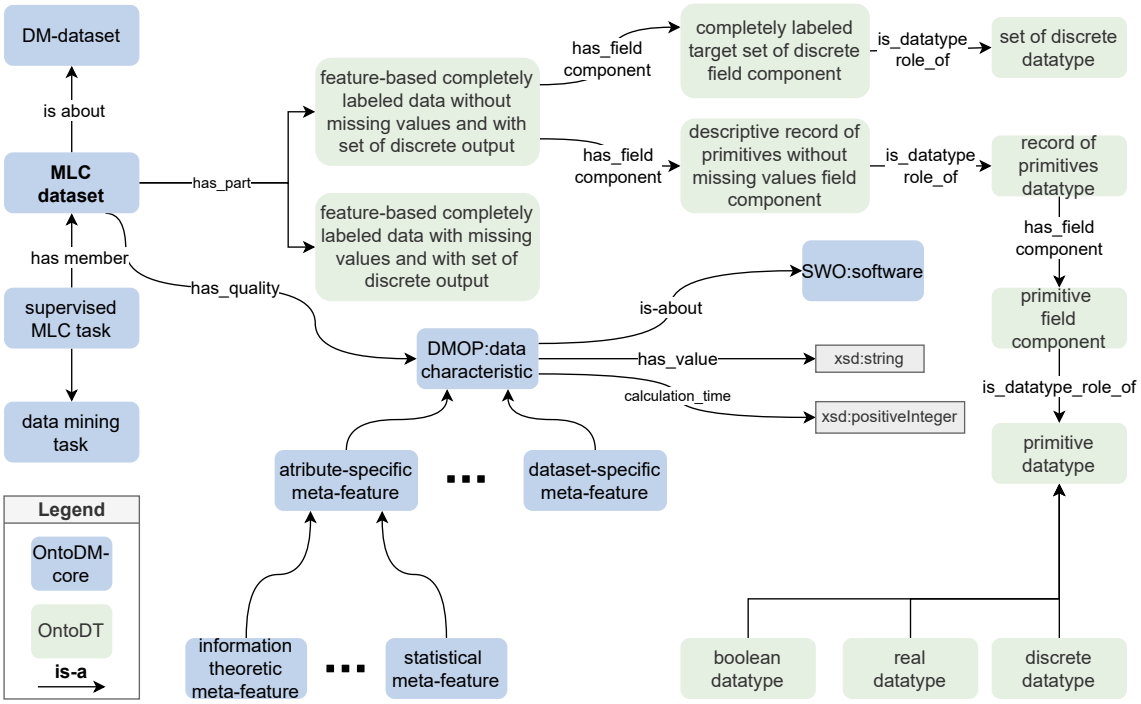


Figure 3.1: ML-specific semantic annotation schema for MLC datasets based on the OntoDM-core [26] and OntoDT [118] ontologies. The schema allows annotation of the different datatypes appearing in the datasets, specification of the data mining task, and representing MLC-specific meta-features as data characteristics.

Figure 3.1 depicts the high-level view of the proposed annotation schema. First, the MLC datasets are represented as instances of the *MLC dataset* class, which in OntoDM-core is modeled as a dataset specification of feature-based data [26]. In order to explicitly encode the learning task, which in our case is MLC, the *MLC dataset* class is connected with the *supervised MLC task* via the `has-part` relation.

To represent the datatypes, we reuse classes from the OntoDT ontology. For example, in the case when the data examples do not contain missing values for the descriptive features, we reuse the *feature-based completely labeled dataset without missing values and with a set of discrete output* class. Each data example is composed of two components, i.e., a descriptive component that contains the descriptive features and a target component for the target labels. For each of the components, there is a corresponding datatype. Then, the datatypes are refined until a primitive (boolean, discrete, real) datatype is reached. A more detailed description of the taxonomy of datatypes, their use in the context of machine learning, and their representation can be found in Panov et al. (2016) [118].

To annotate the meta-features of MLC datasets, we introduce ontology classes into the MLC semantic annotation schema and directly align them with the *OntoDM-core* ontology. In line with the OBO Foundry's guidelines for ontology development, we prioritized reusing classes from existing ontologies. We model meta-features as inherent qualities of a dataset, employing the *data characteristic* class from the DMOP ontology [119] to facilitate this representation. By employing the `is about` property from the Information Artifact Ontology (IAO), we link these data characteristics (i.e., meta-features) to the software libraries that implement them.

Table 3.4: The list of MLC meta-features included in the MLC semantic annotation scheme.

	MetaFeature	isSubClassOf	Library
1	Default accuracy	data characteristic	MULAN [120]
2	Ratio test to power		
3	Ratio total to power		
4	Ratio train to power		
5	Ratio unseen to test		
6	Total distinct classes		
7	UnseenInTrain		
8	Attributes	data characteristic/ dataset-specific meta-feature	MLDA [9]
9	Distinct labelsets		
10	Instances		
11	Labels		
12	LxIxF		
13	Number of binary attributes		
14	Number of nominal attributes		
15	Number of numeric attributes		
16	Proportion of binary attributes		
17	Proportion of maxim label combination (PMax)		
18	Proportion of nominal attributes		
19	Proportion of numeric attributes		
20	Proportion of numeric attributes with outliers		
21	Proportion of unique label combination (PUniq)		
22	Ratio of number of instances to the number of attributes		
23	Cardinality	data characteristic/ label-specific meta-feature/ label distribution meta-feature/ general meta-feature for labels distribution	MLDA [9]
24	Density		
25	Kurtosis cardinality		
26	Maximal entropy of labels		
27	Mean of entropies of labels		
28	Minimal entropy of labels		
29	Skewness cardinality		
30	Standard deviation of label cardinality		
31	CVIR inter class	data characteristic/label-specific meta-feature/ label distribution meta-feature/label imbalance meta-feature/inter class	MLDA
32	Max IR inter class		
33	Mean of IR inter class		
34	Max IR intra class	data characteristic/label-specific meta-feature/ label distribution meta-feature/label imbalance meta-feature/intra class	MLDA [9]
35	Max IR per labelset		
36	Mean of IR intra class		
37	Mean of IR per labelset		
38	Mean of standard deviation of IR intra class		
39	Average examples per labelset	data characteristic/ label-specific meta-feature/ labels relation meta-feature	MLDA [9]
40	Bound		
41	Diversity		
42	Mean examples per labelset		
43	Number of labelsets up to 10 examples		
44	Number of labelsets up to 2 examples		
45	Number of labelsets up to 5 examples		
46	Number of labelsets up to 50 examples		
47	Number of unconditionally dependent label pairs by chi-square test		
48	Number of unique labelsets		
49	Proportion of distinct labelsets		
50	Ratio of number of labelsets up to 10 examples		
51	Ratio of number of labelsets up to 2 examples		
52	Ratio of number of labelsets up to 5 examples		
53	Ratio of number of labelsets up to 50 examples		
54	Ratio of unconditionally dependent label pairs by chi-square test		
55	SCUMBLE		
56	Standard deviation of examples per labelset		
57	Average gain ratio	data characteristic/attribute-specific meta-feature/information theoretic meta-feature	MLDA [9]
58	Mean of entropies of nominal attributes		
59	Average absolute correlation of numeric attributes	data characteristic/attribute-specific meta-feature/statistical meta-feature	MLDA [9]
60	Mean of kurtosis		
61	Mean of mean of numeric attributes		
62	Mean of skewness of numeric attributes		
63	Mean of standard deviation of numeric attributes		

In this work, we consider a total of 63 MLC meta-features, with 56 implemented in the MLDA library [9] and seven in MULAN library [120]. For the meta-features derived from MULAN, we build a taxonomy as proposed in Bogatinovski et al. (2022) [44]. The remaining seven meta-features are categorized directly as subclasses of the *DMOP:data characteristic* class. The full list of the MLC meta-features and the taxonomy is shown in Table 3.4. For further information on the meta-features, we direct the reader to the respective libraries.

Finally, each meta-feature can be associated with its specific value and the computation time, measured in milliseconds.

In Figure 3.2, we showcase a semantic annotation example of the Birds dataset [121]. The figure illustrates the procedure by which the tabular Birds dataset, along with its metadata files, is processed and annotated using our proposed MLC semantic annotation schema. The outcome, an RDF graph, is displayed at the bottom of Figure 3.2 in the RDF/XML serialization format.

3.3.2 Semantic annotation of MLC experiment and performance data

For semantic annotation of MLC experiment data, we adopt the OntoExp schema for semantic annotation of predictive modeling experiments proposed by Tolovski et al. [122]. The schema covers two DM experiment execution workflows: the classical train-test split experiment execution workflow and the N-fold cross validation (CV) workflow. This schema is aligned with the OntoDM-core ontology, which simplifies the integration with our schema for annotation of MLC datasets outlined in Section 3.3.1, as both are based on the OntoDM-core ontology.

To describe the workflows, OntoExp incorporates essential components of DM experiments (see Figure 3.3), such as DM datasets, which assume different roles (i.e., train or test) depending on their usage, and data folds, which denote the data split in folds in the CV evaluation. These folds originate from the DM dataset used to divide the data into separate splits for the CV process.

Additionally, OntoExp describes the predictive model train test evaluation workflow execution as composed of three parts of processes: the execution of a predictive modeling algorithm (resulting in a trained model), the application of this model to a test dataset (producing predictions and assigning them a dataset with predicted set role), and the calculation of evaluation metrics (in both train-test and N-fold CV workflows) that serve as performance indicators of the trained predictive model.

Building on OntoExp as a foundation, we adapt and expand upon it to develop an ontology-based schema, as illustrated in Figure 3.3, specifically designed for our use case of annotating MLC experiments. The schema ensures that each stage of the MLC experiment, from data preparation through to model evaluation, is semantically annotated, allowing for detailed representation and analysis of the experimental processes and outcomes.

In Table 3.5, we outline the description logic axioms of our OntoExp extension (also highlighted in Figure 3.3). First, we introduce the *Sampling technique* class to annotate details about the process of *DM dataset sampling*. We model the *Sampling technique* class as a subclass of *Scientific technique* (DL1). The *DM dataset sampling* process takes as input a *DM dataset* and outputs a sampled data also represented as an instance of the *DM dataset* class. The input and output relations are modeled with the `has_specified_input` and `has_specified_output` object properties, respectively, as defined in the OBI ontology [123] (DL2). As the sampling process can produce various versions of a dataset, distinguished by different data splits, we track the connection to the original dataset through the *originated_from* object property. This property is applied to the *DM dataset* class in

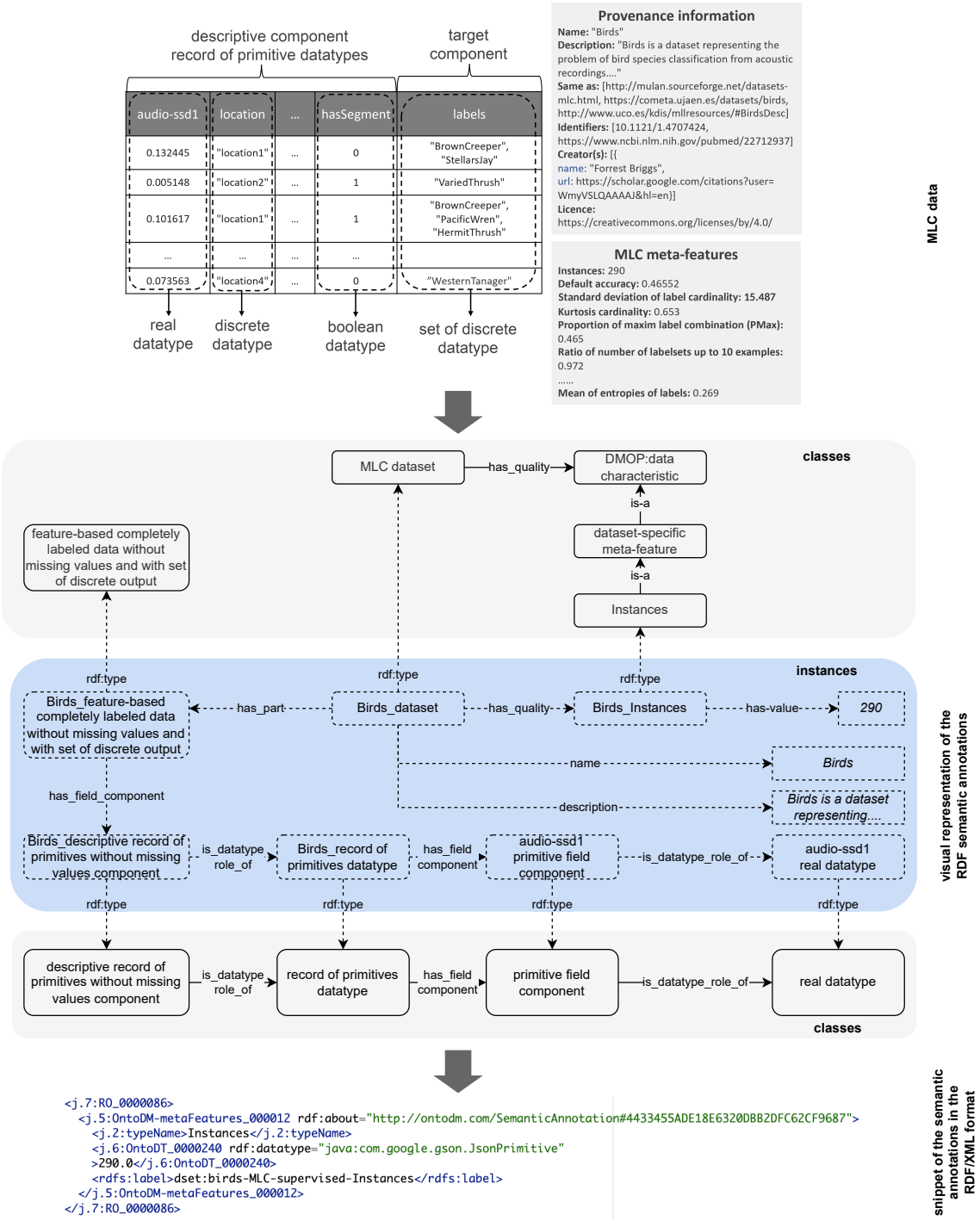


Figure 3.2: An illustrative example of semantic annotation of the Birds dataset [121].

a recursive manner, ensuring that each sampled dataset is linked to a single source dataset by enforcing a cardinality constraint of one (DL3).

We further introduce the *CV train test dataset assignment*, which is used to represent the process of assigning the train and test data in each CV iteration based on the set of fold. The outcomes of this process are DM datasets, each assigned a specific role as either training or testing data. Subsequently, the workflow continues to training and testing the model on these assigned datasets within the respective CV iteration (DL4).

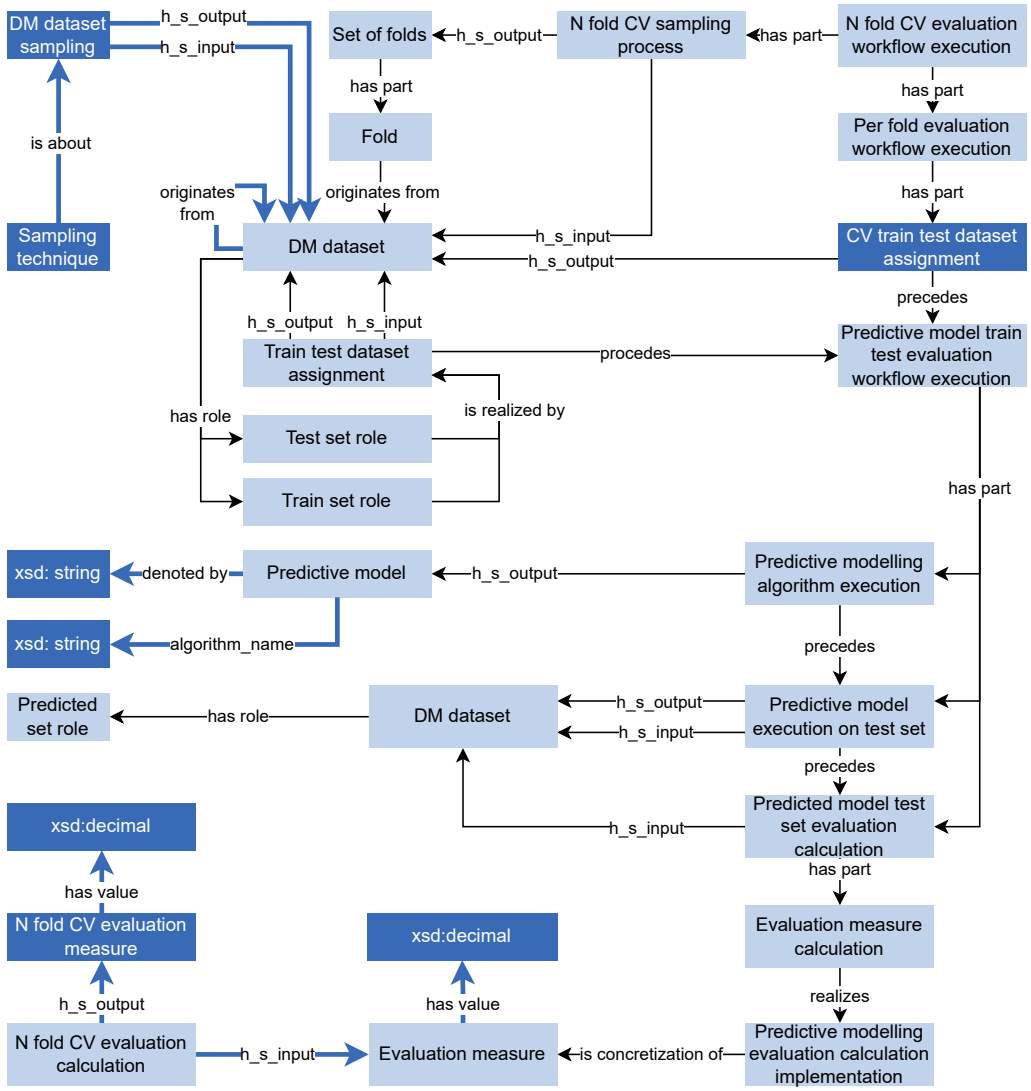


Figure 3.3: An overview of the schema for semantic annotation of MLC experiments and performance data. The novel components in the schema, introduced in this dissertation, are highlighted in dark blue. The properties labeled as 'h_s_output' and 'h_s_input' denote 'has_specified_output' and 'has_specified_input' object properties, respectively.

Furthermore, to represent details about the predictive models, including model hyperparameter - value pairs, we link the *Predictive model* class with the `denoted_by` data property which has string as a range and a cardinality one. Additionally, we utilize the `algorithm_name` data property, also with a cardinality of one and a string range, to easily access the name of the algorithm used for training the model (DL5).

The *Evaluation measure* and *N fold CV evaluation measure* classes are both modeled as sub-classes of the *data item* class with instances of these classes required to have exactly one decimal value (DL6, DL7). *N fold CV evaluation measure* is an new class we introduce, designed to encapsulate an aggregated evaluation measure derived from multiple folds. To facilitate the annotation of the process for calculating the mean aggregated evaluation measure across multiple folds, we use the *N fold CV evaluation calculation* class. This class is structured to accept evaluation measures as input and produce *N fold CV evaluation measure* as its output (DL8).

Table 3.5: Description logic axioms for extending the schema for semantic annotation of MLC experiments and performance data. The special characters used are: $\sqsubseteq$ denotes 'subclass of', $\sqcap$ represents 'intersection' or logical AND, $\exists$ indicates 'there exists' (existential quantifier), $\forall$ represents 'for all' (universal quantifier), and =N specifies a 'cardinality constraint' where exactly N relationship must exist.

	Class	Description Logic
(DL1)	Sampling technique	Sampling_technique $\sqsubseteq$ Scientific_technique $\sqcap$ $\exists$ is_about.DM_dataset_sampling
(DL2)	DM dataset sampling	DM_dataset_sampling $\sqsubseteq \exists$ has_specified_input.DM_dataset $\sqcap$ $\exists$ has_specified_output.DM_dataset
(DL3)	DM dataset	DM_dataset $\sqsubseteq$ =1 originates_from.DM_dataset
(DL4)	CV train test dataset assignment	CV_train_test_dataset_assignment $\sqsubseteq$ $\exists$ has_specified_output.DM_dataset $\sqcap$ $\exists$ precedes.Predictive model train test evaluation workflow execution
(DL5)	Predictive model	Predictive_model $\sqsubseteq$ =1 denoted_by.xsd:string $\sqcap$ =1 algorithm_name.xsd:string
(DL6)	Evaluation measure	Evaluation_measure $\sqsubseteq$ data_item $\sqcap$ =1 has_value.xsd:decimal
(DL7)	N fold CV evaluation measure	N_fold_CV_evaluation_measure $\sqsubseteq$ data_item $\sqcap$ =1 has_value.xsd:decimal
(DL8)	N fold CV evaluation calculation	N_fold_CV_evaluation_calculation $\sqsubseteq$ $\exists$ has_specified_input.Evaluation_measure $\sqcap$ $\forall$ has_specified_input.Evaluation_measure $\sqcap$ $\exists$ has_specified_output.N_fold_CV_evaluation_measure $\sqcap$ $\forall$ has_specified_output.N_fold_CV_evaluation_measure

3.4 MLCBench: Semantic Catalogue of MLC Benchmarking Data

The ontology-based semantic schema for MLC benchmarking data that we have proposed facilitates the creation of semantic MLC catalogue – MLCBench, which makes MLC benchmarking data easier to access and reuse. In this section, we showcase the extensive knowledge base of semantic annotations that forms the foundation of the MLCBench catalogue. Additionally, we present the resources we have developed for semantic annotation and storage of the data, as well as tools for online data access and querying and interactive visualizations.

3.4.1 Knowledge base of MLC benchmarking data

Our MLC benchmarking data knowledge base contains semantic annotations for 89 MLC datasets from different application domains, including medicine, bioinformatics, multimedia, and chemistry. For every dataset, we create a dedicated meta-dataset with semantic annotations. These meta-datasets contain provenance details for the corresponding MLC dataset, incorporating terms from the Schema.org Dataset vocabulary [116] and additional ML-specific annotations as outlined in our semantic annotation schema for MLC datasets. They also include information about 63 MLC meta-features, as specified in Table 3.4. The computation of these meta-features is performed using two Java libraries: MLDA [9] and MULAN [120].

In addition to the datasets, the knowledge base includes annotations about experiments conducted on a subset of the 89 MLC datasets. Specifically, we annotate experiment data about 42 different MLC datasets derived from a comprehensive comparative MLC

experimental study [124]. The study evaluates the performance of 26 MLC methods across these datasets. Each dataset comes with predefined train-test splits. Performance insights are provided at two levels: firstly, from a 3-fold cross-validation phase done on a subset of the training data used to find the best hyper-parameters for the algorithms, and secondly, from the subsequent results that are calculated the predefined train-test data splits using the best performing models as identified in the 3-fold cross validation phase.

This performance evaluation encompasses 18 predictive and two efficiency metrics. The predictive metrics are divided into: six example-based (including hamming loss, accuracy, precision, recall, F1 score, and subset accuracy), eight label-based (such as macro precision, macro recall, macro F1, micro precision, micro recall, micro F1, AUROC, and AUPRC), and four ranking-based measures (one error, coverage, ranking loss, and average precision). Efficiency metrics cover training and testing times. This set of evaluation metrics offers a comprehensive view of each MLC method's effectiveness and efficiency.

3.4.2 System for semantic annotation, storage and querying

The MLC catalogue of bookmarking data is supported by a system that automatically generates and stores semantic annotations and facilitates the execution of semantic queries. The system, powered by the extensive use of Semantic Web technologies has the following key functionalities: (i) a semantic annotation pipeline for MLC benchmarking data, which includes annotation of datasets, experiment workflows and performance data, (ii) a solution for storing these annotations in a RDF triple store; (iii) a REST API that facilitates querying the annotations via SPARQL; and (iv) an online access point to the catalog for MLC benchmarking data, designed to simplify data access, enable effective querying, and support interactive analysis and visualization.

3.4.2.1 System overview

Figure 3.4 provides an overview of the system architecture supporting the MLCBench catalogue. The system is designed to process different MLC data input files, including the MLC datasets in the Weka's `ARFF` format [125], dataset-specific metadata in JSON format detailing provenance information and calculated MLC meta-features, and performance data in tabular (CSV) format. Additionally, the system processes the files that define the semantics, including ontologies, semantic schemas and vocabularies.

The processed data goes through an RDF triplification process, a method for generating semantic annotations that converts the processed input files into a series of RDF triples (comprising subject, predicate, and object components).These RDF triples form a connected RDF graph, containing the input data enriched with semantics. For RDF triplification, we employ the Apache Jena RDF API [126] from the Apache Jena library[127]. Apache Jena is a powerful, open-source Java library for building Semantic Web and Linked Data applications. It has a comprehensive API for handling RDF data, provides robust querying capabilities with SPARQL, and its support for reasoning over data.

The RDF triples thus created are then stored in a triple store. Our architecture incorporates two different triple stores: Apache Jena Fuseki [128], used for storing RDF triples related to MLC datasets, and Virtuoso [28], used for RDF triples related to MLC experiments and performance data. We chose the Apache Jena triple store due to its ease of use and compatibility with the Apache Jena framework. However, the RDF graph containing our semantic annotations for experiments is significantly larger than that for datasets, encompassing a total of 22,908,626 highly interconnected triples. This size difference resulted in slower query execution times with Fuseki, prompting us to explore other triple stores

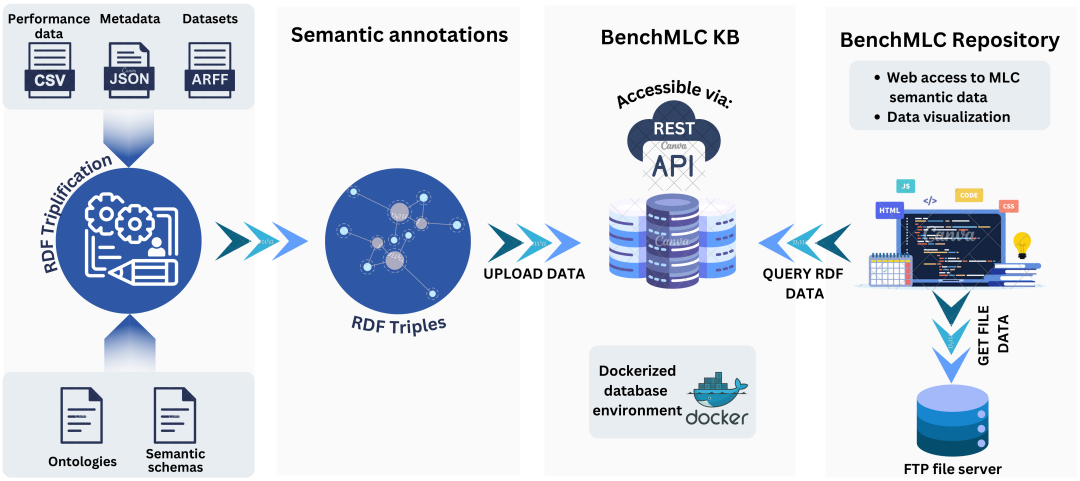


Figure 3.4: A schematic representation of the system architecture for the ontology-based catalog of MLC benchmarking data showcasing functionalities such as semantic annotation processing of MLC benchmarking data, RDF triple storage, REST API for SPARQL query access, and an interactive online catalog for user-friendly data exploration and visualization.

designed to efficiently handle large datasets and complex queries. We chose Virtuoso as our triple store due to its high performance, scalability, and robust support for SPARQL querying. Virtuoso excels in handling large volumes of RDF data and offers efficient query execution. All data within this triple store constitute the MLCBench knowledge base. Alongside the RDF annotations, in both triple stores we store the inferred versions of ontologies to speed up the execution of the queries that require reasoning. The inference is made using the OWL Micro reasoner [129]. It's important to note that having two separate triple stores does not hinder data interoperability. Given that both adhere to the same semantic model, federated SPARQL queries enable seamless simultaneous access to data across these distinct endpoints. This ensures that data can be integrated and queried together, despite being stored in separate storage systems.

To ensure ease of deployment and scalability, the entire semantic annotation storage and querying environment is containerized using Docker. This approach offers several advantages, including simplified configuration, environment consistency, portable workloads and ability to scale the system as needed.

Access to the knowledge base is made easy through a REST API, enabling users to directly submit SPARQL queries. Additionally, we have created a website for the MLCBench catalog, which lets researchers in the MLC field easily find the data they need and explore it in different ways.

Finally, MLC datasets in the `ARFF` format, which have licenses permitting redistribution, are stored on a file server and can be accessed upon request via the FTP protocol. The calculated MLC meta-features, while included in the semantic annotations, are also downloadable in `JSON` format. Additionally, we provide a dump of all RDF annotations on the FTP file server. We make all semantic annotations openly accessible, published under the `https://creativecommons.org/licenses/by/4.0/` license.

3.4.2.2 Querying the knowledge base

To facilitate queries to the knowledge base, we have established a REST API (query endpoint). For example, to query data related to MLC datasets using SPARQL, the following endpoint should be utilized:

`http://semantichub.ijs.si/fuseki/MLC-datasets/query?[insert query]`

In Figure 3.5, we present the SPARQL query for the following query expressed in natural language: "*List all dataset that have Cardinality greater than 2*". Additionally, Figure 3.5 displays the results of executing this query. Cardinality, an MLC meta-feature, represents the average number of labels per data instance. Analysis reveals that, among the 89 MLC datasets cataloged in the knowledge base, 36 have cardinality greater 2.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX OntoDM: <http://www.ontodm.com/OntoDM-core/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX OntoDT: <http://www.ontodm.com/OntoDT#>
PREFIX sa: <http://ontodm.com/SemanticAnnotation#>

SELECT DISTINCT ?datasetLabel
WHERE {
    ?dsetInstance rdf:type/rdfs:subClassOf OntoDM:OntoDM_000144 .
    ?dsetInstance rdfs:label ?datasetLabel .

    ?dsetInstance obo:RO_0000086 ?mfInstance .
    ?mfInstance OntoDT:OntoDT_0000240 ?mfValue .
    ?mfInstance sa:typeName ?mfLabel .
    FILTER(?mfLabel = 'Cardinality' && xsd:float(?mfValue) >= 2) .
}
ORDER BY LCASE(?datasetLabel)
```

	datasetLabel
1	"dset:ABPM-MLC-supervised"
2	"dset:bibtex-MLC-supervised"
3	"dset:bookmarks-MLC-supervised"
4	"dset:cal500-MLC-supervised"
5	"dset:CHD_49-MLC-supervised"
6	"dset:corel16k001-MLC-supervised"
7	"dset:corel16k002-MLC-supervised"
8	"dset:corel16k003-MLC-supervised"
9	"dset:corel16k004-MLC-supervised"
10	"dset:corel16k005-MLC-supervised"

Showing 1 to 10 of 36 entries

First Previous 1 2 3 4 Next Last

Figure 3.5: An example SPARQL query (left) for querying the MLCBench knowledge base and the first 10 answers obtained (right).

Additionally, data related to the benchmarking study is stored on a Virtuoso server. These datasets can be queried via the following SPARQL endpoint: `http://mlcbenchmark.ijs.si:8890/sparql/`.

3.4.2.3 Online catalogue of MLC benchmarking data

Querying the MLCBench catalogue requires familiarity with SPARQL and the underlying semantic model of the knowledge base, presenting a challenge for those less experienced with the Semantic Web technologies. To address this, we have developed a web-based platform (available at `http://semantichub.ijs.si/MLCBench/`) with a simple user interface that simplifies access and interaction with our catalogue of MLC benchmarking data. This online resource effectively abstracts the complexities of ontology-based annotations, enabling query operations within the knowledge base without the need for SPARQL query expertise.

The online catalogue offers several key functionalities. It supports querying based on user-defined parameters like domain, unit of analysis, and metadata text searches (see Figure 3.6). Users can also filter datasets by the value ranges of the MLC meta-feature, by number of descriptive and target features or by the presence of missing values. It also includes a visualization feature that allows users to dynamically explore MLC meta-features and conduct comparisons across various datasets.

Additionally, the MLCBench catalog facilitates in-depth exploration of individual MLC datasets. For example, Figure 3.6 showcases the proteins_virus dataset on the left-hand

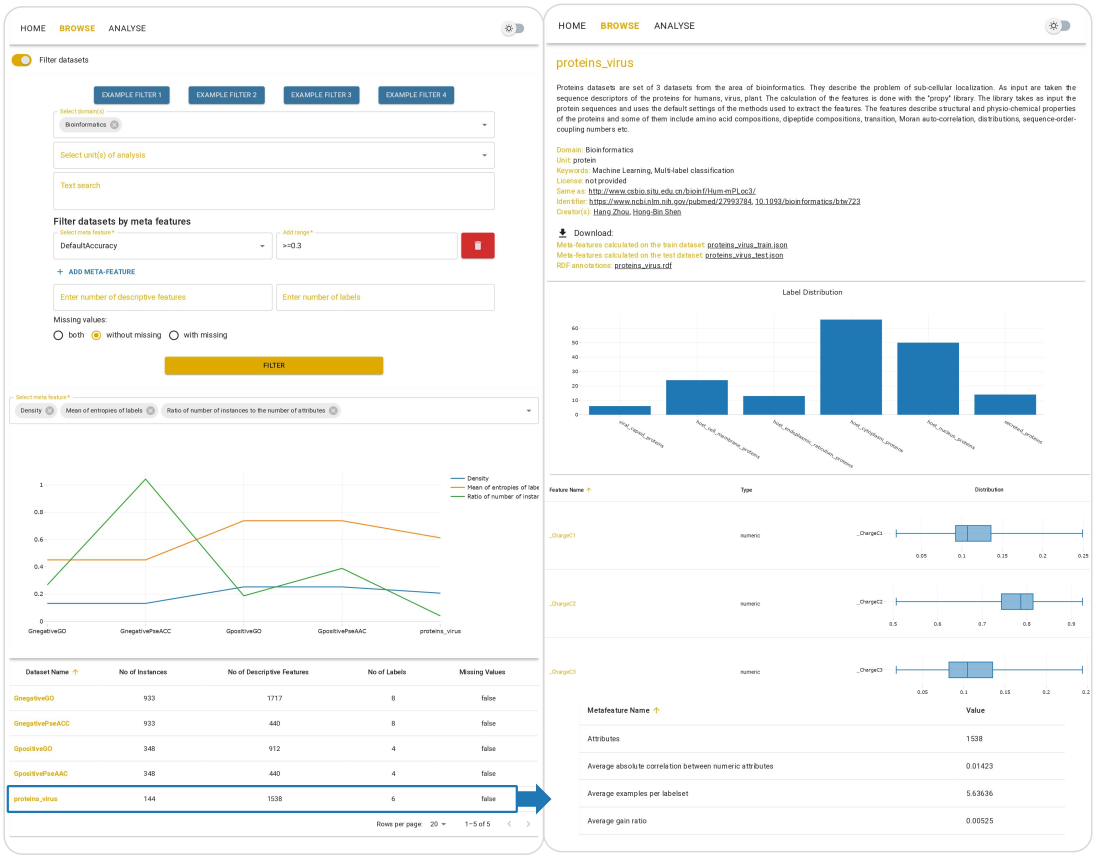


Figure 3.6: A view of the MLCBench online catalog interface. The left side showcases the browsing and querying features for MLC datasets, together with an interactive tool for comparing MLC meta-features across different datasets. The right side presents a detailed page for individual MLC datasets, including visualizations of descriptive features and target label distributions, alongside provenance information and a summary of calculated MLC meta-features.

side. It presents the available provenance information and enables the downloading of MLC meta-features, RDF annotations, and datasets as dump files. Visualizations offer insights into label distributions, descriptive feature distributions, and a detailed listing of all MLC meta-features with their calculated values for each dataset, providing a comprehensive understanding of the data.

Furthermore, the catalogue supports browsing and querying of MLC experiments (see Figure 3.7). Queries can be tailored using various criteria, including the evaluation type (train/test or cross-validation), MLC datasets, MLC methods, and specific ranges for selected evaluation measures.

Moreover, the catalog enables interactive analysis of experiment performance data, as depicted in Figure 3.7. The visualization tools we have created facilitate two forms of analysis: a comparative study of MLC method performance across all accessible MLC datasets, and a detailed comparison of MLC method results on an individually chosen dataset. In both scenarios, users can conduct their analysis based on a performance evaluation measure of their choice.

The MLCBench online catalog presents a list of all MLC methods, including their provenance information and links to their original publications. Additionally, it details the

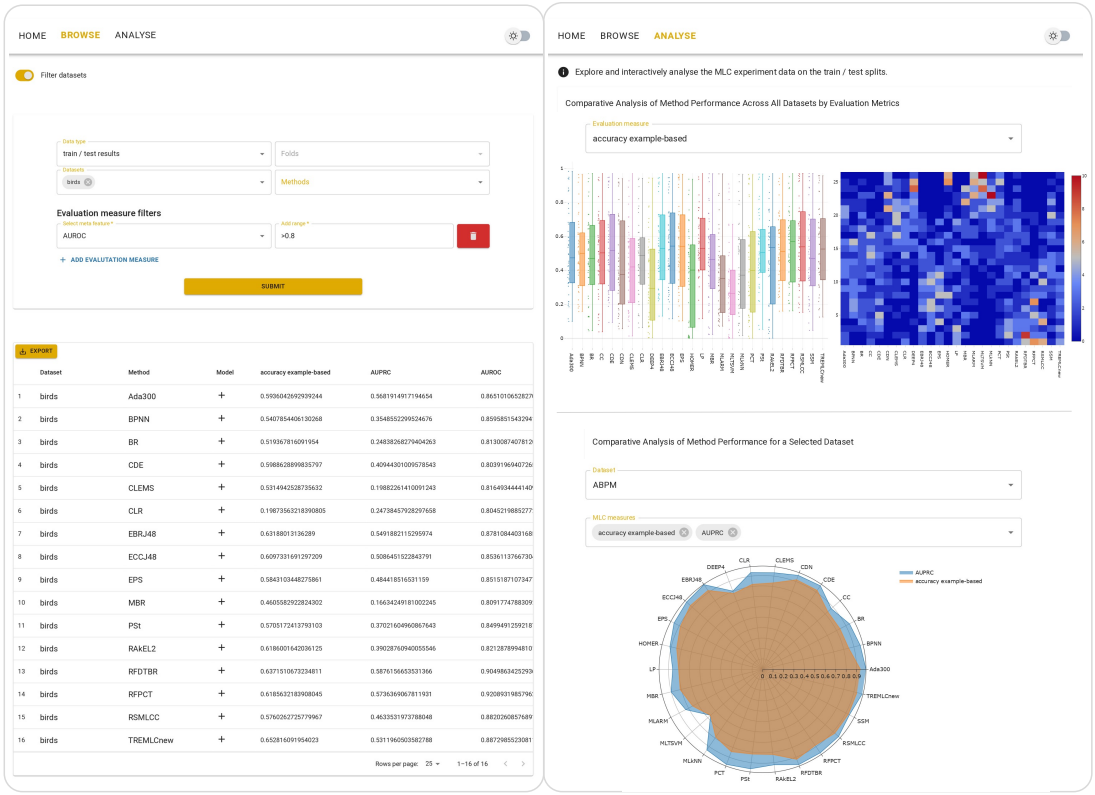


Figure 3.7: A view of the MLCBench online catalog interface. The left side showcases the browsing and querying features for MLC experiments. The right side presents an overview of the interactive visualization tool for analysis of MLC performance data.

available MLC meta-features, providing both descriptions and mathematical definitions for each.

It is important to highlight that MLCBench exclusively utilizes RDF semantic annotations available in the MLCBench knowledge base for its content; thus, all information displayed is directly retrieved through querying this knowledge base with automatically constructed SPARQL queries.

3.5 Summary and Discussion

We have developed an ontology-based schema for semantic annotation of MLC data and an online catalogue that makes the benchmarking data FAIR. Our catalogue introduces a key novelty: all data descriptions are enhanced with semantic annotations (metadata) based on terms from ontologies and controlled vocabularies.

The catalogue provides comprehensive descriptions of 89 MLC datasets from multiple application domains, making it, to the best of our knowledge, the most comprehensive repository of publicly accessible MLC benchmark datasets to date. Each dataset in our catalogue is characterized by a set of 63 MLC meta-features, capturing a wide range of measurable properties relevant to the MLC learning tasks within these datasets. Having all the calculated meta-features in one place allows the experts to jointly observe the properties of the learning task across different datasets. The detailed descriptions of the datasets, including their provenance information, enhance the reusability of the data and improve

the overall data understanding.

We also provide the links to the train and test splits of the datasets as used in a comprehensive study of MLC methods [124]. Providing information about the train/test splits is especially important for benchmarking and reproducibility of computational experiments.

Another significant contribution of this work is incorporating experiment workflows into our knowledge base and semantic catalogue. Each experiment is described in detail, covering aspects crucial for reproducibility such as data sampling techniques, data splits used, algorithms applied, specific hyperparameter sets, and experiment outcomes, including performance across 20 different measures. This level of detail not only enhances the reproducibility of experiments, allowing researchers to accurately replicate studies and verify findings, but also facilitates comparative analysis by providing a basis for directly comparing the effectiveness of different MLC methods under consistent conditions. Furthermore, the semantic annotation of experiments and experiment outcomes promotes methodological transparency. The inclusion of detailed experiment workflows in our semantic catalogue embodies the principles of open science, making MLC research more accessible and collaborative.

Another feature of our catalogue as compared to other MLC repositories, is the interactive nature of the catalogue supported by the underlying web-based system. More specifically, the web-based system allows users to interactively inspect all of the available benchmarking data based on the provided semantic annotations including visual inspection the MLC datasets, their meta-features, descriptive and target data distributions and algorithm performance. This interactivity allows for a deeper and more intuitive understanding of the data.

To conclude, the main contribution of our catalogue is the use of a semantic layer for representing standardized, formal descriptions of MLC benchmarking data through the application of formal ontologies. The rich semantic annotations provide the catalogue with advanced querying capabilities that employ the reasoning power of ontologies. Furthermore, the explicit inclusion of semantics further broadens the range of applications of the available data, as this helps practitioners better understand, reuse and augment the data automatically. Finally, the uniqueness we provide along various dimensions makes our catalogue the go-to source for future benchmarking and evaluation of MLC methods.

Chapter 4

Representation of BBO Benchmarking Data

In this chapter, we introduce OPTION (OPTImization algorithm benchmarking ONtology), an ontology specifically developed for the semantic representation of BBO benchmarking data. We start the chapter with the problem definition and motivation for doing this study. This is followed by an overview of related work in the field. Next, we focus on the ontology itself, detailing its design and implementation, as well as the core entities that make up the ontology. We also provide several use cases to illustrate how the ontology can be utilized for the annotation of BBO benchmarking data. Finally, we present the OPTION system, which is designed for annotating, storing, and querying data. This includes a discussion on the integration of the OPTION Knowledge Base (KB) within the IOHprofiler environment as well as a discussion on the future extension of the ontology and its knowledge base.

This chapter is based on the article "OPTION: OPTImization Algorithm Benchmarking ONtology" [130], [131], published in IEEE Transactions on Evolutionary Computation and at the Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) Companion, 2021.

All data, code, and resources developed for this chapter are publicly available on GitHub at: `https://github.com/KostovskaAna/OPTION-Ontology`.

4.1 Problem Definition

Evolutionary computation (EC) and black-box optimization (BBO) in general are fast-growing fields that have made tremendous progress recently. Due to the numerous applications in engineering, artificial intelligence, and beyond, new optimization algorithms are constantly being developed, making it impossible for researchers and practitioners in the field to keep up with all the new developments. On the other hand, data sharing has gained significant acceptance in recent years. Nowadays, it is de facto standard to publish research results and data on publicly accessible data repositories, whenever possible, to promote their reusability. However, while data sharing undoubtedly helps to achieve this aim, there are unfortunately no common standards for *which* data to record, nor *how* to store it. Consequently, the storage, sharing, and reusability of benchmark optimization data is challenging because different data formats are only partially compatible. In the context of numerical optimization, for example, there are several important benchmarking tools, each with its way of storing performance data, such as COCO [57], Nevergrad [56], and SOS platform [132] . While each of these individual data formats is internally con-

sistent, they handle the details of data storage differently. Interoperability of data from different sources is therefore limited unless explicit conversions are made. Management of benchmark data is even more challenging if we consider the "Reproducibility guidelines for AI research" [133], [134], where various ACM reproducibility concepts are presented: i) repeatability (same team, same experimental design), ii) reproducibility (different team, same experimental design), and iii) replicability (different team, different experimental design). These guidelines are also discussed in the context of EC [135].

While performance data is often made available (with limited interoperability) via benchmarking platforms, detailed information about the algorithms generating this data is usually lacking. This is in part due to the complexities inherent in describing optimization heuristics. Even within a single family of algorithms, differences in operator choices, parameter adjustment strategies, and hyper-parameter settings can result in very different algorithm behavior. If these design decisions can be stored in combination with the corresponding performance data, this would open the door to extracting knowledge from the vast amount of data generated every day.

Besides performance data and algorithm descriptors, data on the problem landscape are crucial in benchmark studies. Yet, the computation of landscape features often demands significant computational resources. To prevent the need for repeated computation, it is important to ensure that this data is both reusable and interoperable.

In summary, we see an increasing amount of data that could be used to select or even design algorithms in an informed way. However, we also see increasing complexity in identifying and processing relevant data. This is largely due to various challenges in the EC domain for data integration and interoperability and it often leads to biased comparisons and reduced transferability of knowledge.

4.1.1 Domain challenges for data integration and interoperability

There exist many different benchmarking platforms for optimization, each with their own way of storing performance and algorithm data. Three main approaches to the storage of performance data are described below:

- **Csv-based**: The data is stored as a single file per experiment in a csv-based format, where each column represents a performance measure or other meta-information. An example is the format used in Nevergrad [56]. This allows for storing data on many different functions/problems into a single file, with the drawback that the granularity of the data is often limited.

- **Textfile-based**: The data is separated into a single file per function/problem, where the meta-information is delimited in some way, followed by the performance information. This format is easily extendable and human-readable, but it can be hard to work with when files become large. An example of this format is used by the SOS platform [132].

- **COCO/IOH-like**: The data is separated into multiple files and folders: generally, folder structure splits along algorithms and functions/problems. Each folder then contains a file with meta-information about the runs, with links to the files where the raw performance data is stored. This structure makes it easy to find the data sought, but the different links to the files can be an obstacle for practitioners who are not used to this format. Variants of this data format are used by COCO [57] and IOHprofiler [136].

As mentioned, each of these data formats has its advantages and disadvantages. While there are some commonalities between different methods, the particularities in handling

meta-data make interoperability of the data from different sources challenging. Furthermore, these differences lead to more limited post-processing functionalities available to the users of these platforms since they are only compatible with those tools that support their particular data format. While these tools are slowly becoming more interoperable, this process requires significant effort from the developers of the individual tools to make sure all data formats are fully supported. A common data structure would be useful to the benchmarking community to avoid each developer having to do this individually.

Additional source of complexity in recording performance data from black-box optimization is that we typically do not use a single performance measure. Instead, we are interested in analyzing algorithm performance from different perspectives: small vs. large budgets, the time needed to identify solutions that meet specific quality criteria, the robustness of the algorithm in search and performance space, etc. [5].

To enable such detailed analyses, researchers often record performance data in a multi-dimensional fashion, spanning at least the time elapsed (measured in terms of CPU time and/or function evaluations), solution quality, and robustness. We may also be interested in how dynamic parameters evolve during the optimization process, in which case we record their values along with the performance data. Both requirements add another level of complexity to the data formats and may explain why they differ so much in practice.

Several other factors further contribute to the complexities surrounding the interoperability and re-usability of publicly available performance data from different benchmarking experiments. We discuss these in the remainder of this section.

Most black-box optimization algorithms are, in fact, families of various algorithm instances. They can be selected by specifying the (hyper-)parameters of the algorithm and/or the operators (e.g., one may speak of Bayesian optimization regardless of the internal optimization algorithm that is used to search the surrogate model, or one may use different acquisition functions, different techniques to build the surrogate, etc.). Different configurations can lead to drastically different search behaviors (and hence performance), and it is crucial to associate the recorded data with the appropriate algorithm instance, not just the algorithm family. However, this is not an easy task, as it can happen that essentially the same algorithm is published under different names. The reader can consult [137], [138] for recent examples and a discussion, respectively.

A similar issue can appear on the problem side. Different instances of the same problem can be of different complexity, and it is not always clear which problem instances were used within a given benchmark study. In addition, some benchmarking suites automatically rotate, shift, permute, or translate the problem instances, to test problem characteristics dependent on those transformations and the generalizability of the algorithms. Other suites do not do this (e.g., because the variable order or absolute values carry some meaning) but still refer to problems of different complexity under the same name. As for the algorithms, we can also have the same problem appear under different, possibly multiple, names. The OneMax problem, for example, is sometimes called CountingOnes, OnesMax, the Hamming distance problem, or Mastermind with 2 colors. All these names refer to the same problem.

Identifying such issues cannot (as of yet) be done automatically but requires human expertise to annotate the data correctly. While this requires a significant amount of effort for the large amounts of currently available benchmarking data, we aim for the procedure to convert from different data formats to be automated where possible (e.g., by involving the authors of the different benchmarking platforms) and clearly structured where not. In the future, this would then become second nature when introducing a new algorithm / problem / experimental setup, allowing the data ontology to grow organically. The creation of reproducible and readily available data will eventually benefit the optimization

Table 4.1: OPTION ontology competency questions.

N^0	Competency question
1	Which problem instances belong to a given benchmark problem?
2	What is the provenance data related to a given benchmark study?
3	Which algorithms are benchmarked in a given study?
3	Which specific operators or hyperparameters are used in a given algorithm, and how are they configured?
4	What are the values of ELA features of a given problem instance calculated on a sample obtained by using a given sampling technique?
5	What was the fitness achieved for a given benchmark problem after a fixed amount of function evaluations?
6	How many function evaluations were needed to reach a given fitness target?
7	Which algorithm(s) achieve the best performance given a fixed number of function evaluations?

community as a whole, so the efforts invested to achieve this goal would be very much worthwhile.

4.1.2 Addressing data integration challenges with ontologies

To develop ontology-based solutions for the integration of benchmarking performance and problem landscape data from different data sources, we need an ontology that formalizes the knowledge in the domain of interest. The ontology should cover the competency questions presented in Table 4.1.

Once the ontology is defined, it can be used by different ontology-based systems and/or benchmark platforms as a common vocabulary for semantic annotation of the data. Benchmark platforms can keep their proprietary data format. As long as they annotate the data with semantic metadata and store the annotations in a semantic data store compliant with the proposed ontology, the data would be automatically interoperable with other knowledge bases and platforms that follow the same protocol for data management.

One of the goals of this dissertation is to design an ontology for semantic annotation of benchmark performance and problem landscape data as well as to design a prototype data management system that enables data integration and reusability with the use of the ontology as a common vocabulary. This goal is operationalized as a set of requirements that an ontology-based system for data integration should fulfill. The requirements are presented in Table 4.2.

To achieve our goal, as outlined by the competency questions in Table 4.1, we have developed the OPTION ontology. Its utility is showcased in an ontology-based data management system in Section 4.4, which aligns with the requirements specified in Table 4.2.

4.2 Related Work

Several efforts have been made to conceptualize different aspects of domain knowledge about EC. The Evolutionary Computation Ontology has been developed to model the relations between algorithm settings (i.e., solution encoding, operators, selection, and fitness evaluation) and different types of problems [139]. It is focused on describing the properties of algorithms, which can be especially helpful for teaching EA-related topics. In the domain of multi-objective optimization, the Diversity-Oriented Optimization Ontology has

Table 4.2: Requirements of the ontology-based system.

N^0	Requirement
1	Semantically annotate benchmarking performance data, problem landscape data, and algorithm configuration details from different benchmark platforms and different problem test suites with ontology-defined terms.
2	Store semantic annotations in a specialized semantic data store.
3	Load and query benchmark performance data from experiments performed using the same or a different system/platform.
4	Load and query problem landscape data for benchmark problems defined in the same or different test suite.
5	Load and query provenance information associated with the benchmark studies.
6	Allow members of the community to extend the ontological conceptual model to cover parts of the domain knowledge missing in the latest active version of the ontology.
7	Allow members of the community to upload their performance data and problem landscape data to extend the system's knowledge base.

been developed, including a taxonomy of algorithms concerning the diversity concept in different search operators [140]. Complementary to the diversity concept, the Preference-based Multi-Objective Ontology has also been proposed to model the knowledge about preference-based multi-objective evolutionary algorithms [141].

The above ontologies have a strong focus on specifics, resulting in classifications of algorithms that allow users to ask only about high-level relations. For example, finding algorithms that use a specific type of operator, finding algorithms that can solve problems from a particular class, and finding algorithms that have been applied to a specific engineering problem. What is missing are ontologies that add semantics to available benchmark data, so that high-level relations and conclusions can be drawn from the ontologies.

4.3 The OPTION Ontology

We developed the OPTION ontology with the primary goal of formalizing knowledge about benchmarking optimization algorithms, emphasizing the formal representation of data from the performance and problem landscape space.

Additionally, we aimed to formally represent details about the optimization algorithms, including aspects like such operators and hyperparameters. Given the variety of optimization algorithms available, our effort in this formalization process is concentrated specifically on modular optimization algorithms. In this context, we demonstrated a proof-of-concept by representing two modular frameworks. While our ontology is easily adaptable for representing various other modular algorithms, it's important to acknowledge that all existing algorithms proposed outside of modular frameworks is a time-consuming and challenging task. This requires the participation of the entire community to reach a consensus on the standard unified representation of black-box optimization algorithms. Addressing this comprehensive task is beyond the scope of the dissertation.

Thus, OPTION offers a comprehensive description of the domain covering the benchmarking process and the core entities involved in the process, such as optimization algorithms, benchmark problems, problem landscape properties, and performance evaluation measures. The ontology currently covers the domain of continuous optimization, but the

classes are defined in a way to be easily extended in the future.

4.3.1 Ontology design and implementation

The design of the ontology was governed by the competency questions listed in Table 4.1. In the ontology design phase, we followed best practices of ontology engineering, i.e., the OBO Foundry principles [10], which ensure interoperability with other external ontologies that follow the same design principles. Our proposed ontology adheres to the single inheritance principle, does not contain any orphan classes, and heavily reuses formally defined relations from the Relations Ontology (RO) [142]. Furthermore, we aligned OPTION with Basic Formal Ontology (BFO) [143], a widely-used upper-level ontology, which served as a template to organize the class hierarchy. The ontology is also aligned with mid-level ontologies, such as the Information Artifact Ontology (IAO)[1] and Ontology of Biomedical Investigations (OBI) [123]. Finally, we reused classes from external ontologies, such as the Generic ontology of datatypes (OntoDT) [118], and Semanticscience Integrated Ontology (SIO) [144].

The OPTION ontology consists of 403 classes and 4130 axioms, including 591 subClassOf axioms. We used the `rdfs:label` annotation property to provide human-readable English labels. The ontology is implemented as an OWL 2 DL ontology with a $SROIQ(D)$ level of expressiveness.

$SROIQ(D)$ builds upon ALC, which is the foundational description logic that supports basic class definitions, conjunctions, and negations. In $SROIQ(D)$ expressiveness, S is abbreviation for ALC, while R extends it with role chains, including transitivity and role hierarchies. I introduces inverse roles, enabling relationships to be defined in both directions. O refers to nominals, which allow for the representation of specific individuals and their equality or inequality. Q includes qualified cardinality restrictions, and D supports datatype properties.

For the development, we used Protégé [145], an open-source ontology-development and knowledge-acquisition environment. The ontology is publicly accessible via the permanent identifier `http://purl.archive.org/option-benchmarking-ontology` and is also available on BioPortal[2] [146], the largest public repository of ontologies.

4.3.2 Ontology layers

As mentioned above, the domain classes from the OPTION ontology are aligned with middle- (IAO and OBI) and upper-level (BFO) ontologies. Orthogonally to that, at each level (domain, middle, and upper), where appropriate, we implement the specification-implementation-execution ontology design pattern [147]. This pattern helps us describe the different aspects of one concept: For an optimization algorithm, we define three conceptually different classes, i.e., *optimization algorithm*; *optimization algorithm implementation*; and *optimization algorithm execution* to represent general information about the algorithm; characteristics of its implementation; and information about the process of its execution, respectively (see Figure 4.1). Similarly, the *optimization algorithm benchmark study design execution* and *optimization algorithm execution* classes are modelled using the same design pattern (see Figure 4.1). This representation enables flexibility and contextual relevance, as in some cases, we focus on a specific perspective (e.g., implementation) of the modeled concept, while the rest can be irrelevant.

[1]IAO ontology: `https://github.com/information-artifact-ontology/IAO/`
[2]OPTION at BioPortal: `https://bioportal.bioontology.org/ontologies/OPTION-ONTOLOGY`

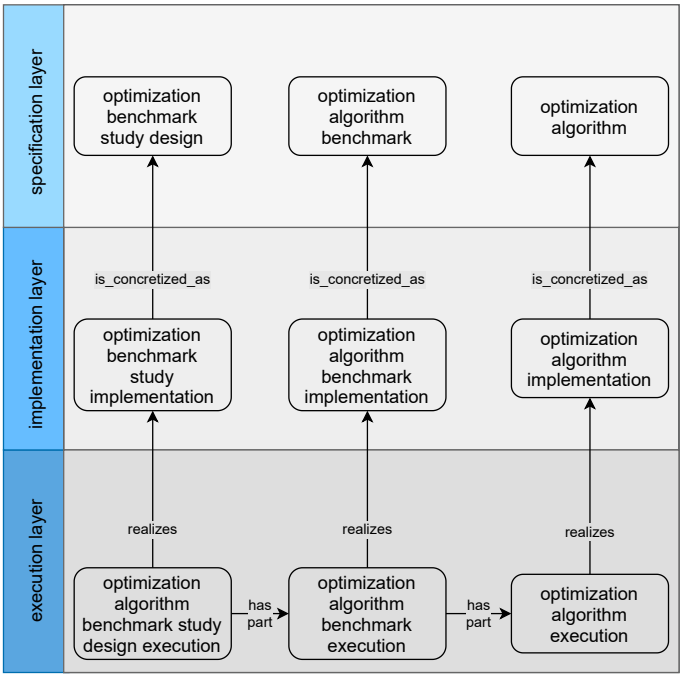


Figure 4.1: The specification-implementation-execution design pattern as used in the OPTION ontology.

4.3.3 Core entities

In this section, we introduce the core entities that make up the foundation of the OPTION ontology and present how they are semantically represented within the ontology.

4.3.3.1 Specification, implementation and execution layers

The OPTION ontology is structured around several objects (i.e., continuants): *benchmark problem*[3], *optimization algorithm*, *function evaluation*, *solution*, *performance evaluation measure* and others, as well as processes (i.e., occurents) in which these entities participate such as *optimization algorithm benchmark study design*, *optimization algorithm benchmark execution*, *experiment run*, and *function evaluation run*. The notion of continuants and occurents comes from the BFO top-level ontology. More specifically, BFO divides all classes/entities into those two disjoint categories. Subclasses of the *continuant* class are objects (including information artifacts), while subclasses of the *occurent* class are processes as they can be extended through time.

4.3.3.2 Semantic representation of core entities

We will briefly describe how optimization entities are semantically defined within the OPTION ontology.

For a visual exploration of the ontology classes, we refer the reader to: `https://service.tib.eu/webvowl/#iri=https://raw.githubusercontent.com/KostovskaAna/OPTION-Ontology/main/OntoOpt.owl`. However, to explore the ontology fully (not just classes but also axioms), we advise the reader to load the OPTION ontology

[3]In the remainder of this paper, we will refer to the ontology classes in OPTION in *italic* font, while the relations between the classes will be written in `typewriter` font.

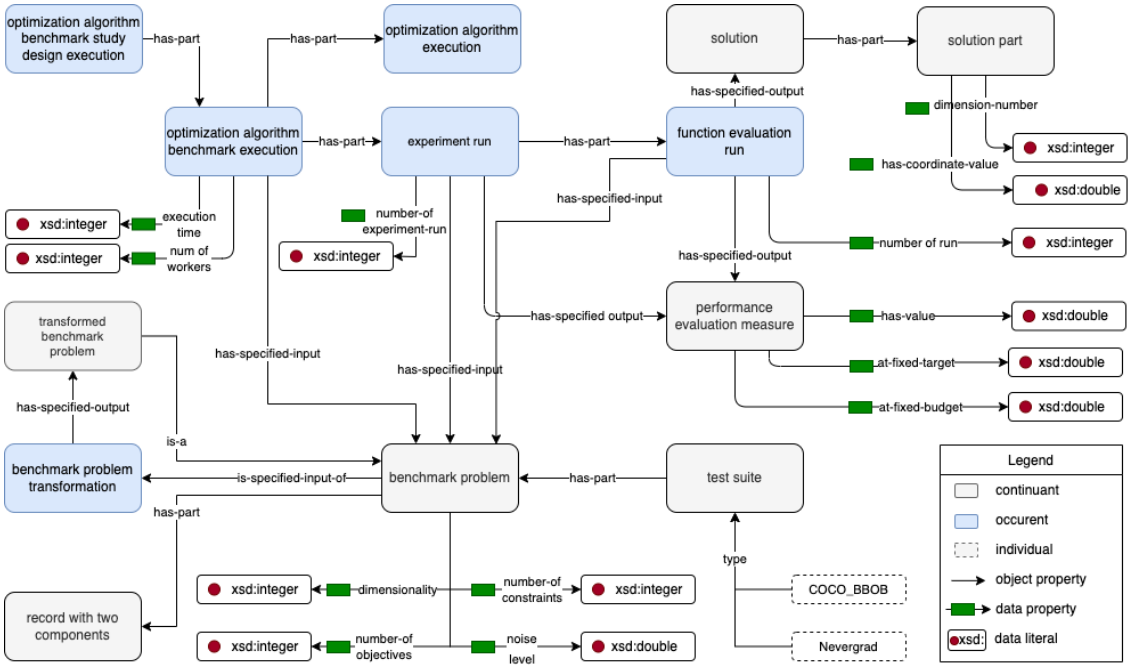


Figure 4.2: The core entities in the OPTION ontology and their relations.

(the .owl file) in Protégé [148]. The structure of the OPTION core entities in the ontology and their relations are presented in Figure 4.2.

The *benchmark problem* class is part of a test suite and it can undergo a process of *benchmark problem transformation* (e.g., shift and scale). The transformed benchmark problem inherits all the properties of a benchmark problem. Thus, it is represented as its subclass in the ontology via the `is-a` property. For each benchmark problem at instance level we can define data properties such as `dimensionality`, `number-of-objectives`, `number-of-constraints`, and `noise level`. For the representation of datatypes on the decision and objective space, we imported the *record with two components* class from OntoDT. The first component is associated with datatypes in the decision space and the second with datatypes in the objective space. Since different test suites can have different benchmark problems or variations of the same, we use the *benchmark problem* class as a root class to build the taxonomy of benchmark problems for each test suite separately. For example, 24 benchmark problems constitute the taxonomy of BBOB benchmark problems.

To represent the study design and study design execution concepts in the context of benchmarking optimization algorithms, we defined the *optimization algorithm benchmark study design* and *optimization algorithm benchmark study design execution* classes as specializations of classes already defined in the OBI ontology. Since keeping track of the provenance information related to each study is a very important aspect in the context of reproducibility and provenance of experiments, we imported a number of properties from the well-known Dublin Core [149] vocabulary and metadata schema, such as `dc:identifier`, `dc:title`, `dc:date`, and `dc:creator`, to name a few.

In one study, we can benchmark a set of optimization algorithms. This relation is captured with the `has-part` transitive object property between the *optimization algorithm benchmark study design execution* and *optimization algorithm benchmark execution*, which represents the execution of each individual algorithm (see Figure 4.2).

The *optimization algorithm benchmark execution* class includes the specification of the input(s) (i.e., the benchmark problem) and its sub-processes. The execution process is

composed of two sub-processes: *optimization algorithm execution* and *experiment run*. Here, we also specify details about the execution, such as the execution time and the number of workers (when parallelization is allowed).

The definition of the experiment run class includes specifics about the input(s) of the process (i.e., the benchmark problems) and output(s) (i.e., performance evaluation measure) of the execution process. In the ontology, various performance evaluation measures have been represented. These include the measured fitness, the best-measured fitness, and the noise-free fitness measure. In addition, the ontology also supports the representation of performance data at the level of function evaluations. For that purpose, we use the same *performance evaluation measure* as presented above. Note that we associate information about the function evaluation runs where applicable, as not all benchmark data is given at this level of granularity. Finally, performance can be measured in a fixed-target or fixed-budget scenario.

Moreover, we also define a *solution* as an output of each *function evaluation run* process. Each solution is broken down into multiple parts, and for each solution part, we represent its location via the `has-coordinate-value` data property. The number of solution parts depends on the dimensionality of the problem.

4.3.4 Representation of problem landscape entities

The problem landscape space is represented with ELA features. *ELA features* in the OPTION ontology are defined as *data items* (see Figure 4.3). They are linked with the corresponding *benchmark problem* via the `is-about` relation. The *benchmark problem* class in Figure 4.2 is the same one as in Figure 4.3 and it connects the two figures. Since the ELA feature value depends on the *sampling technique* and the *sample size*, this information is also included in the ontology. We have already included five sampling techniques that are most common in the literature. However, the list can be extended with other sampling techniques on-demand.

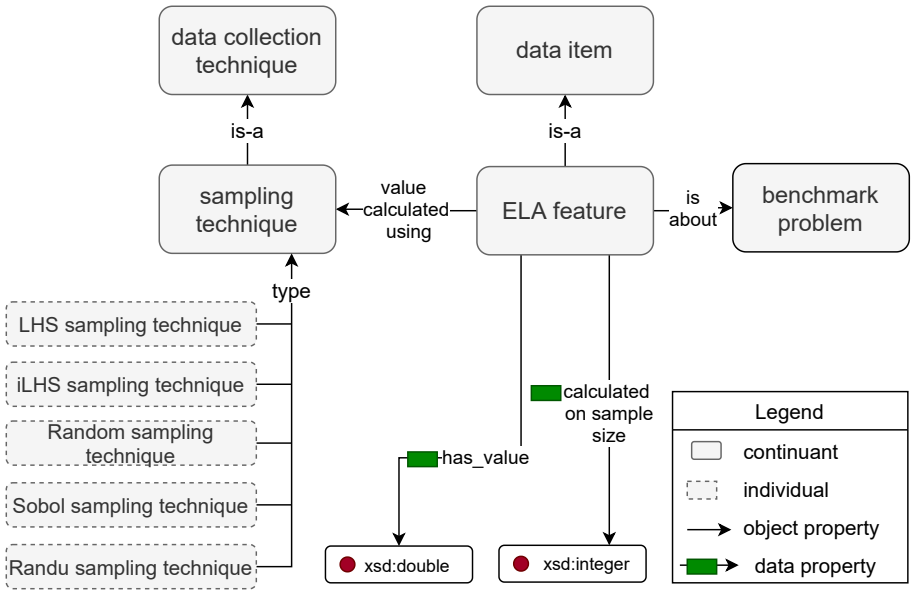


Figure 4.3: Representation of ELA features in the OPTION ontology.

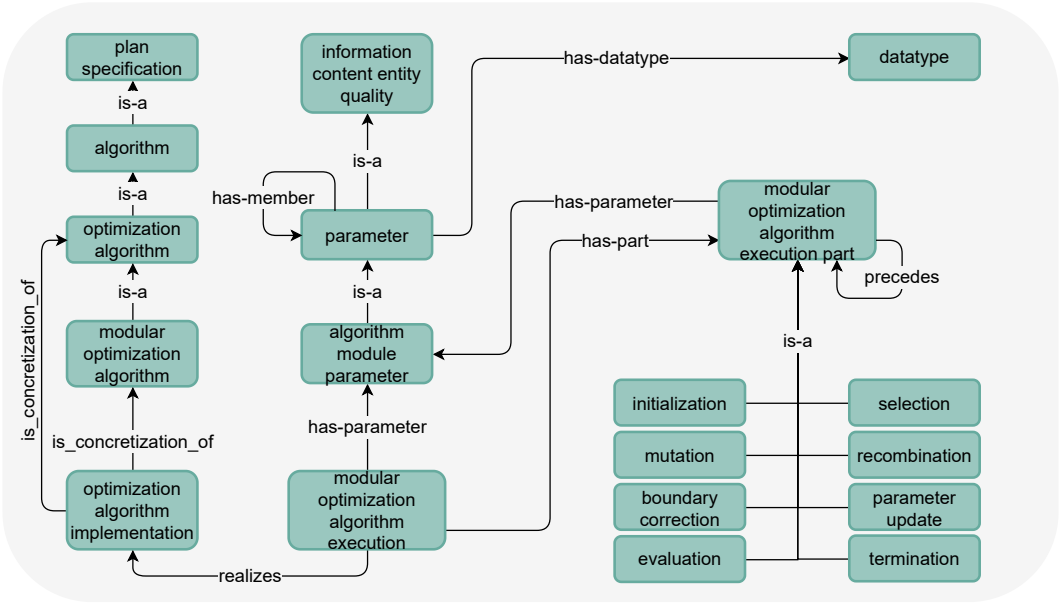


Figure 4.4: The entities and relations for the representation of modular optimization algorithms.

4.3.4.1 Representation of algorithm entities

For the formal representation of modular optimization algorithms, we consider two different families of evolutionary algorithms: Differential Evolution (DE) [76] and Covariance Matrix Adaptation Evolution Strategies (CMA-ES) [77]. Since these two algorithms have been well-researched for over a decade, many variations and modifications have been proposed, resulting in the development of modular frameworks specifically tailored for them.

For the CMA-ES, we use the modCMA framework [78], which contains many variants of the core algorithm. This ranges from modifications of the sampling distributions (including mirrored or orthogonal sampling) to different weighting schemes for recombination to different restart strategies.

For DE, we use the modDE package [79] available at `https://github.com/Dvermetten/ModDE`. This framework provides a wide range of mutation mechanisms, with different modules for selecting the base component, the number of differences included, and the use of an archive for some of the difference components. In addition, the usual crossover mechanisms can be enabled, as well as update mechanisms for internal parameters based on several state-of-the-art DE versions.

For the formal representation of modular optimization algorithms, we have created a separate ontology module within the OPTION framework, that seamlessly integrates with the rest of the OPTION ontology. This ontology module allows us to specify the different steps in the optimization process and link them to the corresponding module parameters (see Figure 4.4). For this purpose, we introduced the *modular optimization algorithm* class as a subclass of the *optimization algorithm* class, which is already defined in OPTION. For modular algorithms, we have also defined a specialized class *modular optimization algorithm execution*. Optimization algorithm execution can be a composition of several subprocesses (e.g., initialization, mutation, and recombination). To model this in the ontology, we have defined the *modular optimization algorithm execution part* class and linked it to the *modular optimization algorithm execution* class via the `has-part` relation. The algorithm execution

flow is represented with the `precedes` relation. *Algorithm module parameters* are linked to both *modular optimization algorithm execution* and *modular algorithm execution part* through the `has-parameter` relation.

4.3.5 Use cases

To demonstrate the benefits of using a common ontology for semantic annotation of data and to address some of the domain challenges for data integration, we consider four different use cases or data sources. The BBOB and Nevergrad benchmark suites are two use cases for the annotation of performance data, a large set of publicly available ELA data is a use case for the annotation of problem landscape data. Finally, we show example annotations of modular optimization algorithms.

4.3.5.1 BBOB

Since 2009, annual workshops have been organized around the benchmarking of derivative-free black-box optimization algorithms with the COCO environment [57]. We consider the BBOB single-objective benchmark suite [150], which consists of 24 single-objective benchmark functions. Some of the data generated during these workshops is freely available [151]. It covers results for the problems of different dimensions $D \in \{2, 3, 5, 10, 20, 40\}$.

We use the BBOB data that includes the algorithms from the 2009-2020 workshops in this use case. The data includes 226 algorithms, which we semantically annotate using the OPTION ontology. For each of the 226 algorithms, semantic annotations in the form of RDF graphs were generated and uploaded to a semantic data store.

A brief overview of the structure of the BBOB benchmark data format was presented in Section 4.1.1. However, to properly annotate the performance data, we need to look at the specifics of the corresponding file formats (see the illustration of raw COCO-BBOB data in Figure 4.5). First, the performance data is indexed by a function evaluation: for each evaluation that improves the objective function, a line gets written in the results file, containing the evaluation number, the raw objective value, and the transformed objective value (i.e., the value that is returned to the algorithm during the optimization process).

The raw objective values are needed to allow for a fair comparison between different instances of the same function, while the transformed values are helpful when trying to reconstruct the input which was given to the optimizer.

In addition to these two values, their best-so-far equivalents are also stored. When dealing with noiseless optimization and only writing data on function improvement, these values are redundant, but they can be helpful in other cases such as noisy optimization.

Not all considered algorithms have data available on the same function/dimension/instance combination. This is partly caused by the shifting requirements of the BBOB workshops; i.e., the set of recommended instances and the number of repetitions per instance have not been identical throughout the years. In addition, some algorithms have been run only on a subset of the available BBOB collection, e.g., because of limited computational resources available. Since most of the algorithms benchmarked on BBOB are stochastic, there is a certain degree of variance between the runs.

The data in the ontology provides the terms/classes needed for annotation on the used problem/dimension/instance/algorithm in the corresponding benchmark analysis. In addition, data provenance information has been manually collected, linked to the performance data to trace its origin, and uploaded in the OPTION KB. The stored data provenance information includes the digital object identifier (DOI) of the paper where the experiments have been presented, the paper's title, the authors' name, and the year of publication. It is therefore possible to filter the data with respect to these criteria.

4.3.5.2 Example annotations of COCO-BBOB performance data

In Figure 4.5, we provide an example of a semantic annotation of performance data for the MLSL algorithm, which was benchmarked on the BBOB test suite using the COCO platform. We illustrate the process of creating OPTION-based semantic annotations in the RDF format.

As previously described, COCO separates performance data into multiple files and folders. More specifically, for each benchmark problem, there is a separate .info text file, which includes meta-information about the runs, and .dat files containing the raw performance data for these runs.

We created a parser for COCO-formatted files and merged the information from the different files into a single table (see the processed raw data table in Figure 4.5).

The processed performance data is then passed to the semantic annotation pipeline, that generates instances (also called individuals) of the OPTION classes. The semantic annotations are saved in the RDF format (which has a graph-like structure), where each fact is expressed in the form of subject - predicate - object triple. The subject and object are represented as nodes in the graph, with the predicate forming an edge between them. For instance, f1_i1_dim2 - rdf:type - f1 is an RDF triple denoting the fact that the problem instance labeled as f1_i1_dim2 is of type f1, where f1 is a class in the OPTION ontology representing the first benchmark problem from the BBOB test suite.

At the bottom of Figure 4.5, a portion of the RDF annotations expressed in the RDF/XML syntax is displayed. Finally, the RDF/XML files are uploaded to a semantic data store (or a triple store) where they can be queried.

4.3.5.3 Example annotations of Nevergrad performance data

The second use case covers semantic annotation of benchmark data obtained from Nevergrad - an open-source platform for black-box optimization [56]. Nevergrad provides different test suites to benchmark the optimization algorithms. In the OPTION KB, we included annotations of 32 optimization algorithms benchmarked on the ten test suites YABBOB, YABIGBBOB, YACONSTRAINEDBBOB, YAHDBBOB, YAHDNOISYBBOB, YAHDSPLITBBOB, YANOISYBBOB, YAPARABBOB, YASMALLBBOB, and YASPLITBBOB. YABBOB [152] is a benchmark suite for black-box optimization problems inspired by the COCO-BBOB test suite. Moreover, in Nevergrad, there are different counterparts of YABBOB. For example, the YANOISYBBOB, YAHDBBOB, YAPARABBOB, YABIGBBOB variants of YABBOB contain problems with noise, high-dimensional, parallel, and big computational resources, respectively. Each of the test suites consists of 21 benchmark problems. The data is publicly available at `https://dl.fbaipublicfiles.com/nevergrad/allxpsnew/list.html`.

There are several differences between the Nevergrad and COCO-BBOB benchmark data. First, the data obtained from the Nevergrad platform is stored at a coarser granularity level. Essentially, only the quality of the final solution is recorded, along with the experimental setup (budget of function evaluations, properties of the problem, etc.), and it is not recorded at the level of each function evaluation as is the case with COCO-BBOB. Also, no provenance data is available for the Nevergrad performance data. On the other hand, Nevergrad records offer other information that is lacking in COCO-BBOB (e.g., number of workers when running function evaluation in parallel and noise level of the function).

The differences in the performance data are inevitable as different benchmark platforms have different data formats. However, it is essential to note that the OPTION ontology was developed with special care not to be biased towards specific benchmark platforms.

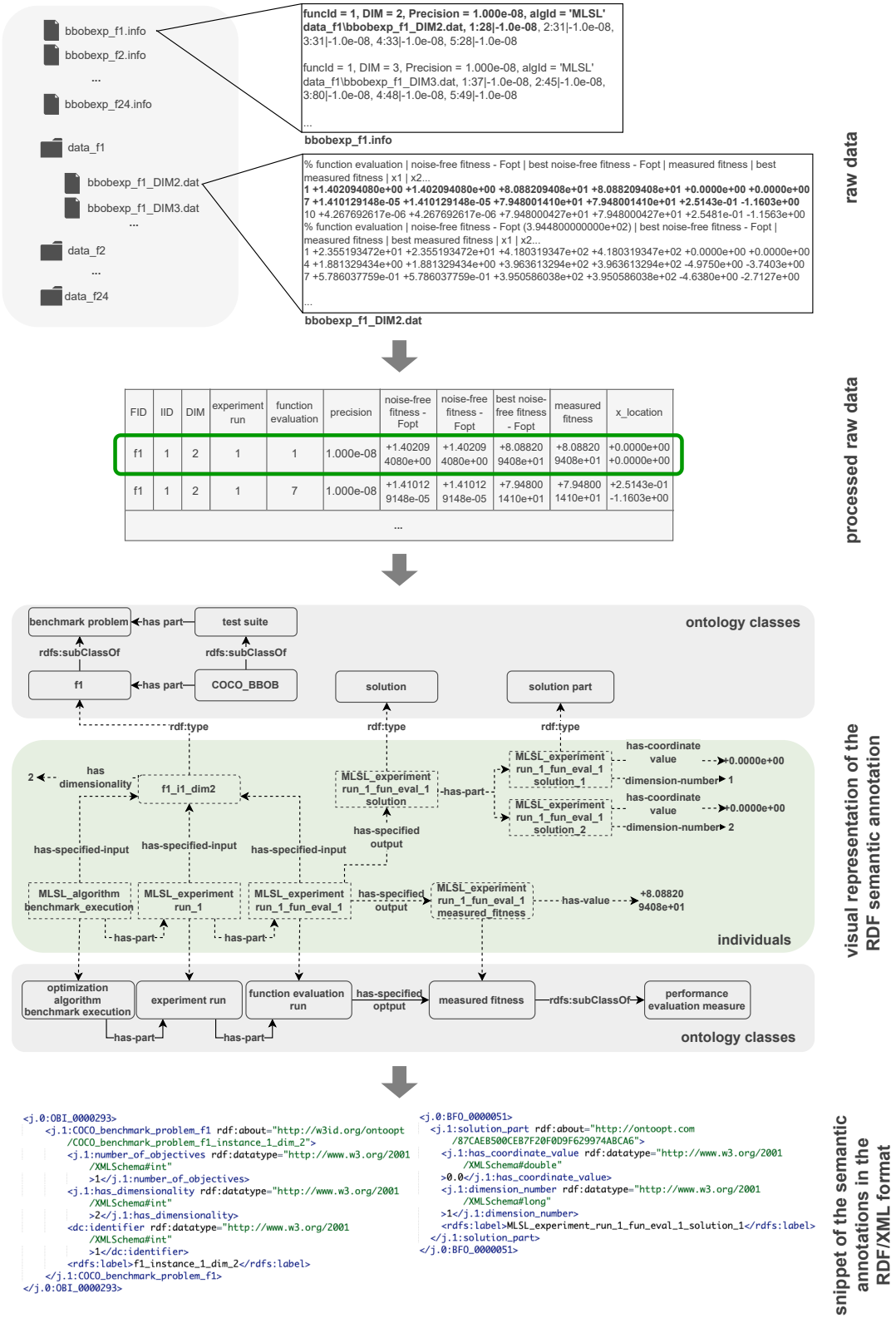


Figure 4.5: An illustrative example of semantic annotation of COCO-BBOB performance data.

The annotation schema we propose is flexible enough to be used for semantic annotation of

benchmark data from various platforms. Indeed, while there may be some information in other platforms that has not been considered while designing the ontology, the annotation schema can be easily extended to cover those aspects without affecting previously annotated data.

4.3.5.4 Example annotations of problem landscape data

In the third use case, we demonstrate the use of the OPTION ontology for the annotation of problem landscape data. For that purpose, we use a publicly available dataset [63] that contains the ELA features calculated for the first five instances of the 24 BBOB noiseless functions from the COCO environment in dimensions $D \in \{5, 10, 15, 20, 25, 30\}$. The 46 ELA features come from six feature groups (dispersion, y-distribution, meta-model, information content, nearest better clustering, and principal component analysis). Since ELA features are not absolute and depend on the sampling strategy and the sample size [153], this information is also added to the knowledge base. In our knowledge base, we include ELA features calculated with five different sampling strategies (i.e., LHS, iLHS, Random, Sobol, Randu) with $30D, 50D, 100D, 250D, 650D, 800D, 1000D$ sample sizes on a total of 100 independent repetitions. In addition, we store the median ELA feature value across these 100 repetitions.

The computation of ELA features is computationally intensive. Having calculated features in a format that automatically links them to the corresponding problems and enables easy access and querying is a large step towards more reusable research.

4.3.5.5 Example annotations of modular algorithms

For the forth use case, we semantically annotate data about modCMA and modDE algorithms. Due to the computational infeasibility of collecting data for all possible combinations of modCMA and modDE algorithms, we opted to use a subset of, specifically 324 algorithm variants for modCMA and 576 variants for modDE. We show the modules and parameter spaces used for CMA-ES and DE in Table 4.3 and Table 4.4, respectively. To obtain the algorithm variants, we created a Cartesian product of the modules and the selected module parameter spaces.

Table 4.3: The complete list of modCMA modules and their respective parameter space yielding a total of 324 algorithm configurations.

Module	Parameter space
Elitist	True, False
Mirrored_sampling	None, mirrored, mirrored pairwise
base_sampler	gaussian, Sobol', halton
weights_option	default, equal, $(1/2)\hat{}\lambda$
local_restart	None, IPOP, BIPOP
step_size_adaptation	csa, psr

In Figure 4.6 we have illustrated the ontological representation of the modDE algorithm. In the ontology, we create specialized subclasses of the general classes corresponding to the modDE versions. For example, the *modDE execution* class is a subclass of *modular optimization algorithm execution*. It inherits all the properties of its superclass but also contains definitions that are unique to the modDE algorithm, such as the different execution parts, their execution order, and links to the modDE module parameters. We note here that in Figure 4.6 only the execution parts such as initialization, mutation, and recombination are shown, while the others (i.e., boundary correction, evaluation, selection, parameter

Table 4.4: The complete list of modDE modules and their respective parameter space yielding a total of 576 algorithm configurations.

Module	Parameter space
mutation_base	rand, best, target
mutation_reference	None, pbest, best, rand
mutation_n_comps	1, 2
use_archive	True, False
crossover	bin, exp
adaptation_method	None, shade, jDE
lpsr	True, False

update, and termination check) have been omitted due to space constraints. Finally, in Figure 4.6, we present two modDE configurations (as instances of the modDE class) that differ by the crossover type, which is a parameter that affects the recombination part of the optimization process. The modeling of the modCMA algorithm is done similarly.

Finally, each of these algorithm variants is linked to performance data. Unlike in the previous use cases where we reuse publically available data, here we generate new performance data. For running the algorithms, we make use of the IOHexperimenter module [154] of the IOHprofiler benchmarking environment [136].

The problem instance portfolio consists of the 24 single-objective black-box optimization problems sourced from the BBOB benchmark suite of the COCO benchmark environment. We consider the first 5 instances of each of the 24 BBOB functions, both with dimensions $D = 5$ and $D = 30$. This results in two separate problem instance portfolios, one for each dimension, with each portfolio containing a total of 120 problem instances.

To evaluate the performance of each algorithm variant, 10 independent runs have been conducted and the median objective function value has been recorded for each problem instance. Our objective function measures the precision of the algorithm's solution, i.e., the distance to the optimum, within a fixed budget of function evaluations. We considered six different budget values, $B \in \{50D, 100D, 300D, 500D, 1\,000D, 1\,500D\}$, where D is the problem dimensionality. We store the best precision achieved by each algorithm variant at the different cut-off budgets for the $5D$ and $30D$ problem instance portfolios. The population size for both CMA-ES and DE is set to $4 + \lfloor 3\log(D) \rfloor$.

Since the problem portfolio consists of BBOB problem instances, this performance data is semantically annotated as described in Section 4.3.5.2.

4.4 The OPTION System for Semantic Data Management

In this section, we describe the OPTION ontology-based system we developed for semantic data management and integration in order to provide means for semantic annotation, storage, and querying of BBO benchmarking data.

The design of the system is governed by the goal and requirements presented in Section 4.1.2. It currently supports the following four components: (i) pipeline for semantic annotation of COCO-BBOB performance and landscape data, Nevergrad performance data; modular algorithm descriptions and performance data (ii) storage of the annotations in a RDF triplestore; (iii) REST API for querying the annotations and integrated query component in the IOHprofiler environment; (iv) web-interface for enabling users to contribute to OPTION and to the OPTION KB and to upload their own COCO-BBOB and Nevergrad data that will be semantically annotated.

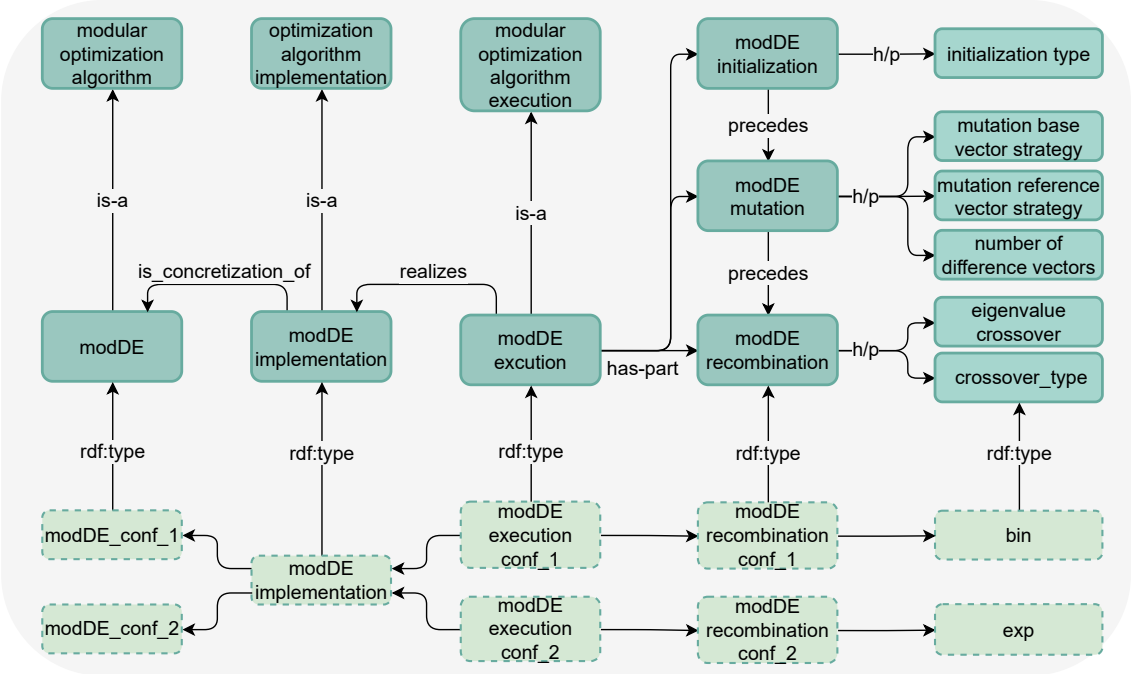


Figure 4.6: An illustration of the representation of the modDE algorithm in the ontology and two examples of annotation of modDE configurations. Rectangular boxes correspond to the ontology classes. Dashed rectangular boxes correspond to the class instances.

4.4.1 The OPTION KB: annotation and storage

The OPTION ontology contains the semantic model, represented in a formal and standardized way. The OPTION KB, on the other hand, leverages the power of the ontology and holds the actual data that has been semantically annotated using the vocabulary of the OPTION ontology. In Section 4.3.5, we discuss four use-cases of OPTION for integration of COCO-BBOB performance, Nevergrad performance, and COCO-BBOB landscape data. For that purpose, we have created three separate KB instances, OPT_BBOB_KB, OPT_Nevergrad_KB, and OPT_ModularAlgo_KB, that comprise the OPTION KB (see Figure 4.7) and store the respective semantically annotated data.

For semantic annotation of the raw data, we developed pipelines to parse the data, and created the semantic annotations using the Apache Jena RDF API.[4] Once the annotation process is completed, the produced RDF annotations are uploaded to the Apache Jena TDB2 triple store.

The BBOB, Nevergrad, and ModularAlgo KB instances are deployed on the same data server. However, that does not prevent other practitioners in the field of EC from creating new KBs hosted on other servers. If they use the same vocabulary, the interoperability between the KBs is assured, meaning that they can be queried simultaneously if the data from multiple KBs is merged.

4.4.2 The OPTION KB: querying semantic annotations

For querying the OPTION KB, we can use the SPARQL query language [155]. We have set up an Apache Jena Fuseki2 server that connects to the Apache Jena TDB2 triple store and implemented two services to enable this functionality. The query service provides

[4]Apache Jena RDF API: `https://jena.apache.org/documentation/rdf/index.html`

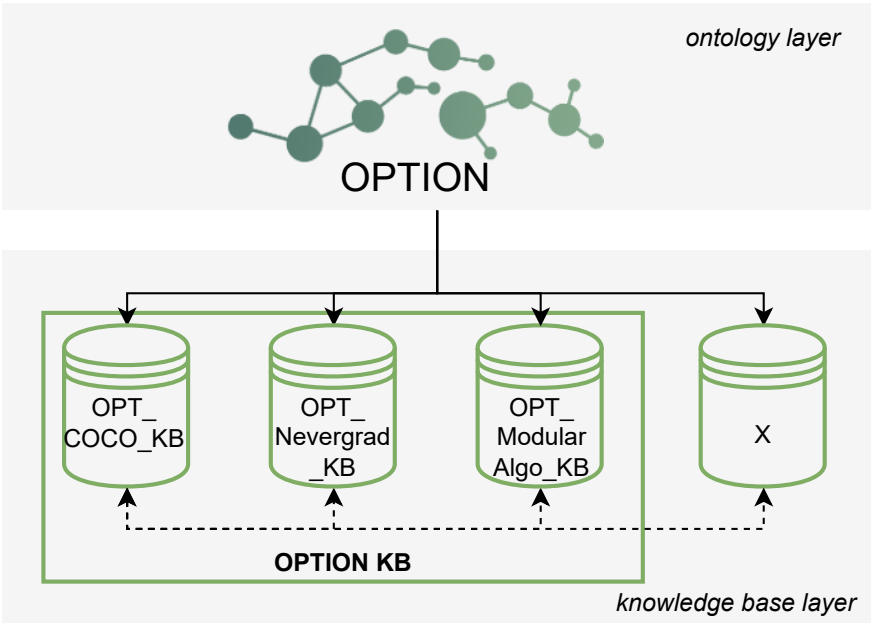


Figure 4.7: The OPTION ontology and the OPTION-aligned knowledge bases. Solid arrows signify the knowledge bases' explicit alignment with the ontology, which is accomplished through semantic annotation of the data. The interoperability of the various knowledge bases is denoted by dashed arrows, which is a direct result of the use of OPTION as a common vocabulary for the annotation of heterogeneous, distributed data.

an endpoint for handling SPARQL queries in a RESTful manner [156], while the upload service enables the upload of RDF data into the triple store.

In Figure 4.8, we present the listing of the SPARQL query for the following query expressed in natural language:

> *For all algorithms included in the study with DOI 10.1145/2739482.2768467 and for a fixed budget scenario with 1000-2000 function evaluations, return the noise-free fitness - Fopt performance evaluation measure calculated on the first five instances of the f1 and f7 benchmark problems from the BBOB benchmark suite.*

In addition, the bottom part of Figure 4.8 shows the first five matches/answers for the same query.

The query service supports all OPTION competency questions (and their combinations), as presented in Table 4.1.

4.4.3 Integration of the OPTION knowledge base with the IOHprofiler environment

As we can observe, SPARQL queries can become very complex and sometimes are seen as a bottleneck to the broader acceptance of Semantic Web technologies. We recognize that SPARQL query construction is an error-prone and time-consuming task that requires expert knowledge of the whole stack of semantic technologies. Even experts find it sometimes challenging to query semantic data since they first must get familiar with the data annotation schemes or the structure of the knowledge base.

To facilitate the use of the OPTION ontology, we provide a simple GUI that can be used to gain access to performance data without needing to write SPARQL queries. Currently,

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?function_evaluation_run_label ?num_function_evaluation_run ?problem ?problem_instance_id  ?dimensionality ?noise_free_fitness
WHERE {
    ?optimization_algorithm_benchmark_execution rdfs:label "10.1145/2739482.2768467_optimization_algorithm_benchmark_execution".
    ?optimization_algorithm_benchmark_execution <http://purl.obolibrary.org/obo/BFO_0000051> ?function_evaluation_run .
    ?function_evaluation_run rdf:type <http://w3id.org/ontoopt/function_evaluation_run> .
    ?function_evaluation_run rdfs:label ?function_evaluation_run_label .
    ?function_evaluation_run <http://purl.obolibrary.org/obo/OBI_0000299> ?performance_measure.
    ?function_evaluation_run <http://w3id.org/ontoopt/number_of_run> ?num_function_evaluation_run .
    filter (?num_function_evaluation_run > 1000 && ?num_function_evaluation_run < 2000).
    ?function_evaluation_run <http://purl.obolibrary.org/obo/OBI_0000293> ?problem.
    ?problem rdf:type ?problem_class .
    filter (?problem_class in (<http://w3id.org/ontoopt/COCO_benchmark_problem_f1>, <http://w3id.org/ontoopt/COCO_benchmark_problem_f7>)).
    ?problem <http://w3id.org/ontoopt/has_dimensionality> ?dimensionality .
    filter (xsd:integer(?dimensionality) in ("5"^^xsd:integer, "10"^^xsd:integer)) .
    ?problem <http://purl.org/dc/elements/1.1/identifier> ?problem_instance_id .
    filter (?problem_instance_id < 6).
    ?performance_measure rdf:type <http://w3id.org/ontoopt/noise-free_fitness_-_Fopt> .
    ?performance_measure <http://w3id.org/ontoopt/has_value> ?noise_free_fitness .
}
```

	function_evaluation_run_label	num_function_evaluation	problem	problem_instance	dimensionality	noise_free_fitness
1	"10.1145/2739482.2768467_CMA-MSR_f1_1_dim10_run1122_fer"	"1122"^^xsd:long	<http://w3id.org/ontoopt/COCO_benchmark_problem_f1_instance_1_dim_10>	"1"^^xsd:int	"10"^^xsd:int	"6.200845621E-4"^^xsd:double
2	"10.1145/2739482.2768467_CMA-MSR_f1_1_dim10_run1258_fer"	"1258"^^xsd:long	<http://w3id.org/ontoopt/COCO_benchmark_problem_f1_instance_1_dim_10>	"1"^^xsd:int	"10"^^xsd:int	"2.324759161E-5"^^xsd:double
3	"10.1145/2739482.2768467_CMA-MSR_f1_1_dim10_run1412_fer"	"1412"^^xsd:long	<http://w3id.org/ontoopt/COCO_benchmark_problem_f1_instance_1_dim_10>	"1"^^xsd:int	"10"^^xsd:int	"1.723421161E-5"^^xsd:double
4	"10.1145/2739482.2768467_CMA-MSR_f1_1_dim10_run1584_fer"	"1584"^^xsd:long	<http://w3id.org/ontoopt/COCO_benchmark_problem_f1_instance_1_dim_10>	"1"^^xsd:int	"10"^^xsd:int	"1.58360497E-6"^^xsd:double
5	"10.1145/2739482.2768467_CMA-MSR_f1_1_dim10_run1778_fer"	"1778"^^xsd:long	<http://w3id.org/ontoopt/COCO_benchmark_problem_f1_instance_1_dim_10>	"1"^^xsd:int	"10"^^xsd:int	"1.801939646E-7"^^xsd:double

Figure 4.8: A screenshot from the FUSEKI query endpoint, presenting an example SPARQL query (at the top) and the first 5 answers to the query (at the bottom).

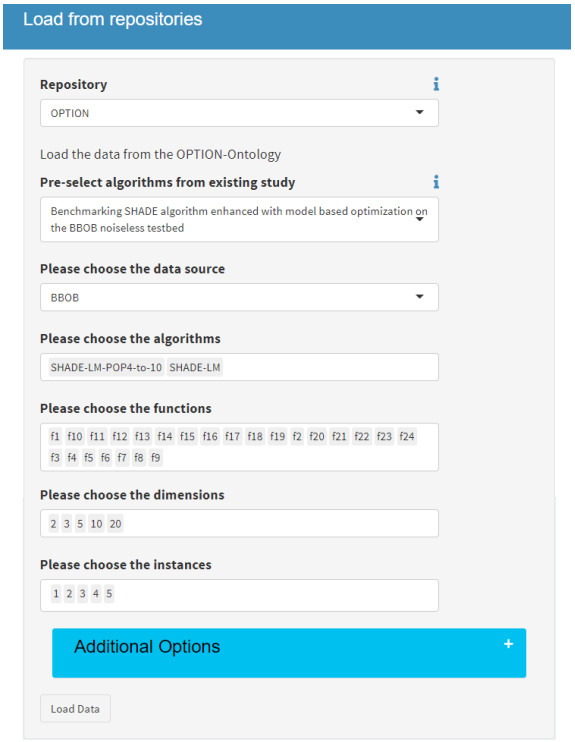


Figure 4.9: The interface of the OPTION-ontology queries within IOHanalyzer (version 1.6.3, available at `https://iohanalyzer.liacs.nl/`.)

the GUI has direct access to the BBOB and Nevergrad KBs, while the modular algorithm data is only accessible via the SPARQL endpoint. This interface is connected directly to IOHanalyzer [157], which enables the loaded data to be used directly in performance analysis and visualization, and even be compared to data that might not yet be included in OPTION or to user-submitter performance data. Furthermore, the GUI provides access to a parameterized search process, which can be used without any underlying knowledge about the used semantic data model. Users can express their query by selecting from several drop-down options, which specify the required information, such as suite, function, algorithm, etc., and load the corresponding performance data to analyze. This interface is shown in Figure 4.9. While this interface is static, it illustrates the power of integrating the ontology into IOHanalyzer: users without any background knowledge can use it to gain insight into the performance of the selected algorithms/functions.

Additionally, this interface can be easily expanded based on the community's wishes. To illustrate this potential, we created another entry point into OPTION, which can be used to load all performance data that originated in a specified paper. To this end, the user selects a paper by its title, which then populates the relevant information about the used algorithms and functions in that study. By loading this pre-selected data, the user has full access to the performance data of the selected study, which they can then investigate in more detail by making use of the visualizations within IOHanalyzer. This type of interactive analysis then allows the user to look at the data from different perspectives and to compare it to other algorithms.

4.4.4 Extending the OPTION ontology and knowledge base

So far, the OPTION ontology has been successfully applied for data integration and management tasks from the COCO and Nevergrad benchmark platforms and two modular frameworks. Data that was previously stored in different data formats and that could not be queried, is now integrated and can be queried simultaneously.

However, extending the ontology is not a trivial task, as it requires its contributors to have a good understanding of the semantic model. Also, the process of annotating performance data from an arbitrary platform, hence populating the knowledge base, currently cannot be fully automated.

To facilitate the uploading of new data to the OPTION KB, we have developed a web interface available at: `http://semantichub.ijs.si/OPTION/`. Currently, the web interface supports the uploading of COCO-BBOB and Nevergrad performance and landscape data from published studies. Figure 4.10 depicts the process of submitting new data, semantically annotating it, integrating the annotations with the OPTION KB, and querying it. Via the web interface, end users can first upload the raw data, details about the study, and related provenance information.

The uploaded data is stored on a Firebase server. Then, periodically, we retrieve the newly uploaded data and semantically annotate it in order to ensure the high quality of the OPTION KB, we include a curator in the loop who prior to executing the semantic annotation pipeline, verifies that the data format conforms to the COCO/Nevergrad data format and verifies the study-related provenance metadata provided by the user.

Finally, requests for extending the OPTION ontology and knowledge base can be made directly via the web interface or via the GitHub repository[5], after which we can establish a collaboration and help guide the whole extension process. We encourage researchers and especially developers of benchmark platforms to adopt the use of the OPTION ontology, to annotate their benchmark and problem landscape data based on the OPTION ontology, and maintain their own OPTION-aligned knowledge bases, as depicted in Figure 4.7. Distributed knowledge bases based on same ontological vocabulary can be easily queried using federated querying strategies[6].

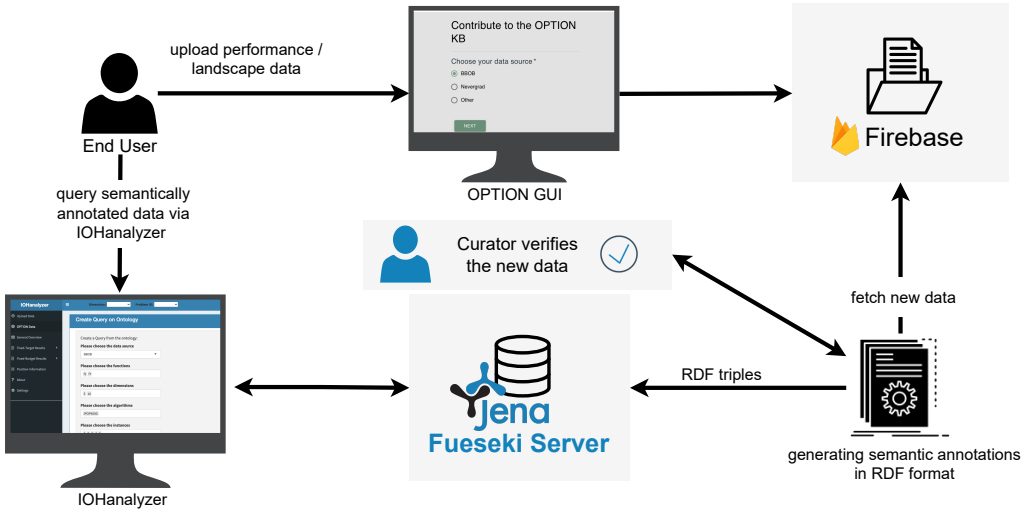


Figure 4.10: A flowchart of the process of uploading, annotating, and querying new data in the OPTION KB.

[5]OPTION at GIT: `https://github.com/KostovskaAna/OPTION-Ontology`
[6]SPARQL 1.1 Federated Query: `https://www.w3.org/TR/sparql11-federated-query/`

4.5 Summary and Discussion

In this chapter, we presented the development of the OPTION ontology, which is specifically designed to address the challenges in managing and integrating BBO benchmarking data. OPTION provides a formal structure for the semantic annotation of BBO performance data, problem landscapes, and algorithm configurations, facilitating data integration across multiple platforms like COCO and Nevergrad.

We demonstrated the effectiveness of OPTION through several use cases: semantic annotation of performance data from the BBOB and Nevergrad test suites, landscape data annotation using Exploratory Landscape Analysis (ELA) features, and the annotation of modular optimization algorithms. Additionally, we introduced the OPTION system, which supports the annotation, storage, and querying of semantically enriched data. This system allows users to query the annotated data without needing to write complex SPARQL queries, thus making the data more accessible.

The current version of the OPTION ontology has been developed from a performance and problem-centered perspective. This perspective allows it to handle the most common types of queries related to benchmark data analysis. Additionally, it covers the representation of modular optimization algorithms. However, this also means that information about the algorithms that are not part of modular frameworks is somewhat limited. The lack of information about algorithms is partly due to the inaccessibility of this type of meta-information: common benchmarking setups only store high-level features about algorithm settings. To further extend the ontology presented in this dissertation, we aim to expand the knowledge base for these algorithm-specific details. Moreover, a more detailed semantic representation of the algorithm space should be included to describe the algorithm family, operators, hyperparameters, etc. Recently, several studies have attempted to unify taxonomies over the algorithm space [158]–[160], but further work is needed to develop a general structure that can be incorporated into OPTION.

The development of the OPTION ontology is a step forward in improving the reusability and interoperability of performance and problem landscape data. By annotating a large subset of BBOB, Nevergrad, and ELA data, we have demonstrated the potential of the ontology to support data integration while providing powerful query capabilities for direct analysis of the required datasets. This significantly reduces the time required to collect data across many functions and algorithms, while providing flexibility in managing the performance perspective (i.e., fixed budget, fixed target).

Chapter 5

Algorithm Selection for Multi-Label Classification

This chapter explores the relatively unexplored area of algorithm selection (AS) for multi-label classification (MLC) ,a significant task in ML that remains largely unexplored in selecting the most effective algorithms for a given MLC problem.. While the preceding two chapters focus on the representation aspects of benchmarking data, this chapter marks the transition toward addressing its exploitation, showcasing practical approaches to leverage benchmarking data for informed decision-making.

The chapter begins by outlining the unique challenges and opportunities associated with applying AS principles to MLC. A review of the relevant literature across various domains provides a foundation for understanding the current landscape of AS research. Subsequently, we elaborate on the ML techniques assessed in this study to tackle AS for MLC. The chapter proceeds with a detailed description of our experimental setup designed to evaluate these techniques. In the results section, we analyze and discuss the performance of each ML approach for AS, emphasizing the explainability of the trained ML models through feature importance analysis. The chapter concludes with a comprehensive summary of our findings and discusses the implications for future research in AS for MLC.

This chapter is based on and extends the paper "Explainable Model-specific Algorithm Selection for Multi-Label Classification", which appeared in the Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), 2022.

All data and code related to this chapter are publicly available on GitHub at: `https://github.com/KostovskaAna/MLC-AS`.

5.1 Problem Definition

In Chapter 3, we have examined the growing availability of MLC algorithms, datasets, and benchmarking data pertinent to MLC tasks and proposed a methodology for improved data management, data interoperability and reusability. However, the large number of MLC algorithms available to tackle MLC problems underscores the important task of AS. AS is driven by the observation that different algorithms exhibit varying performance characteristics across different practical problems. Specifically, an algorithm that excels in certain scenarios may underperform in others. Empirical studies have consistently shown that no single algorithm outperforms others across all datasets in any given ML task [161]. Consequently, the diversity of MLC algorithms introduces a meta-optimization challenge: selecting the optimal algorithm for a new dataset to optimize the performance metric under consideration.

The task of AS is typically addressed using machine learning techniques, which automate the selection process. Automated AS (AAS) methods can replace the tedious and labor-intensive task of manual selection and have already demonstrated promising results in various domains [162]–[165]. A prevalent approach within AAS is the feature-based approach. In the context of MLC, this involves describing each MLC dataset with features that represent the landscape characteristics of the dataset. Using these descriptive features, a supervised machine learning model can be trained to predict the most suitable algorithm for a given dataset.

As discussed in Chapter 3, meta-features tailored to MLC that describe dataset characteristics have been proposed in the literature [9], [120]. In a recent study, Bogatinovski et al. [44] used these meta-features to construct regression models (i.e., multi-target predictive clustering trees) for predicting the performance of three MLC algorithms across five predictive performance metrics. This work not only explored the predictive capabilities but also assessed the meta-features' importance in an unsupervised manner. However, these meta-features have not yet been evaluated in AAS scenarios, revealing a promising area for future research. Additionally, the utilization of AAS in the context of MLC remains largely unexplored. This motivated our study to advance beyond merely predicting algorithm performance. Instead, by training various ML models, we aim to perform AS for MLC, selecting the best-performing algorithm for each dataset individually.

Building on this foundation, our study advances the use of meta-features to predict the best-suited algorithm for individual datasets and introduces an essential element of explainability into the algorithm selection process. Model-specific explanations that clarify decision-making at an instance level are essential, particularly in complex models where understanding the rationale for each decision can significantly enhance trust and usability.

One effective method for achieving this level of explainability is through the use of Shapley values. These values offer detailed local explanations by quantifying the contribution of each meta-feature to the decision-making process for each dataset instance [166]. Shapley values have been widely used as an explainability technique in ML [167], [168]. Unlike classical ML feature importance approaches that provide global importances at the model level, Shapley offers feature importances locally for each prediction. This local interpretability aspect provides valuable insights into the model's decision-making process, enhancing its explainability and potential practical use.

Another prominent open question in AAS pertains to the choice of machine learning approach; it is not clear which approach is best suited for applications in the context of MLC, and whether there is a big difference between the approaches at all. Specifically, the effectiveness of different approaches, such as regression, classification, and pairwise models, in AAS remains underexplored.

In this dissertation, we aim to train ML models to identify the best-performing MLC algorithm for individual datasets. We compare several feature-based supervised ML approaches to AAS and provide an explainability layer that helps us understand how each prediction is made in terms of the importance of different meta-features. In [169], we investigated the influence of ML approaches on BBO AS. In this chapter, we extend this methodology to the MLC domain.

5.2 Related Work

Recent advancements in AS include the introduction of general frameworks that employ meta-learning and ensemble learning techniques, as demonstrated by Tornede et al.[170]. Additionally, Pulatov et al. [92] have adopted a novel, general approach by analyzing source code features of algorithms. This method provides machine learning-based recommenda-

tions for optimal algorithm selection and is versatile enough to be applied across various algorithm types, as long as their source code is available.

In the realm of machine learning, AS has been explored in various contexts. Shawkat and Smith [171] investigated AS in a classification learning scenario involving eight different classifiers and 100 benchmark datasets, while Pise et al. [172] using statistical data descriptors, employed the K-nearest neighbor algorithm to recommend suitable classifiers from a set of thirty-eight benchmark datasets from the UCI repository. Additionally, Cohen-Shapira and Rokach [173] introduced a novel approach for AS in clustering by utilizing supervised graph embeddings.

Within the broader spectrum of ML, various supervised approaches have been used for AS. Predominantly, regression and classification have been foundational in predicting the most suitable algorithms for specific problem instances. For example, regression techniques have been utilized to predict performance metrics directly, as explored by Jankovic et al. [89], while classification methods that directly probide the label of the best algorithm have been detailed by Vškvorc et al. [88]. Additionally, pairwise approaches of regression/classification have also been investigated [91], [92].

Supervised approaches such as regression and classification have been pivotal for AS. Regression methods are used to predict performance metrics, as detailed by Jankovic et al. [89], while classification approaches identify the optimal algorithm, as explored by Vškvorc et al. [88]. Additionally, pairwise regression/classification methods have been investigated to refine AS further, as seen in studies by Van Rijn et al. [91] and Pulatov et al. [92].

Despite the progress in other areas, AS for MLC remains largely unexplored. To the best of our knowledge there are no work that targets directly AS for MLC. This domain presents unique challenges and opportunities for leveraging AS to enhance performance. Accordingly, we aim to develop a landscape-aware AS framework for MLC, employing various supervised ML techniques that utilize meta-features characterizing the distinct datasets and learn to predict the most suited algorithm on a dataset instance level.

5.3 ML Approaches for AS

In this dissertation, we explore several feature-based ML approaches to address the AS problem for MLC. Each approach utilizes a vector of meta-features to describe the characteristics of the MLC datasets. These meta-features serve as inputs to the ML models, and the outputs are the performances or the rankings of the algorithms when applied to these datasets. By employing feature-based ML techniques, we aim to learn a mapping from the input meta-features to the outputs, effectively predicting the most suitable algorithms for a given MLC dataset.

A feature-based ML approach for MLC AS is formally defined as follows: Let $\mathcal{X}$ be a matrix with dimensions $D \times M$, where D represents the number of datasets and M denotes the number of meta-features. Each row $\mathbf{x}_i$ of $\mathcal{X}$ is the meta-feature vector characterizing the i-th dataset. The objective is to train an ML model to learn a mapping function f, which processes the meta-feature vector $\mathbf{x}_i$ and produces an output $y_i = f(\mathbf{x}_i)$. The output y_i represents the prediction for the i-th dataset and could be a continuous value, a class label, or a tuple of continuous values or labels, depending on the ML approach employed.

The following subsections outline the ML approaches investigated in this dissertation.

5.3.1 Regression approach

5.3.1.1 Single-output regression

In the single-output regression approach, we train a separate predictive model f_a for each of the A algorithms in the portfolio, where $a \in \mathcal{A}$. These models, trained independently, predict the performance of their respective algorithms. For the i-th dataset from the dataset portfolio described by its meta-feature vector $\mathbf{x}_i$, the model $f_a(\mathbf{x}_i)$ produces a continuous value prediction $y_{i,a}$ representing the estimated performance (such as accuracy, error rate, or loss) for algorithm a when applied to that dataset.

Once the A models are trained, the performance predictions for each dataset i are compiled into a tuple $(y_{i,1}, y_{i,2}, \ldots, y_{i,A})$. The algorithm associated with the optimal predicted performance for dataset i is selected based on:

$$s(i) = \begin{cases} \arg\max_{a \in \mathcal{A}} y_{i_a} & \text{when the goal is to maximize a performance metric,} \\ \arg\min_{a \in \mathcal{A}} y_{i_a} & \text{when the goal is to minimize a performance metric.} \end{cases}$$

5.3.1.2 Multi-output regression

We also explore a variant known as multi-output regression, where a single model f is trained to simultaneously predict the performance for $a \in \mathcal{A}$ algorithms using the same meta-feature vector $\mathbf{x}_i$. The output $\mathbf{y}_i = f(\mathbf{x}_i)$ directly yields a tuple containing the performance predictions for all algorithms in the portfolio. This approach captures potential correlations between the performances of different algorithms on the same dataset. The selection of the optimal algorithm follows the same criterion as in the single-output scenario.

5.3.2 Pairwise regression approach

The pairwise regression approach extends the basic regression approach by predicting the performance difference between pairs of algorithms for each dataset. This approach can be implemented as either single-output or multi-output regression.

5.3.2.1 Single-output pairwise regression

In the single-output pairwise regression, we train a model $f_{a,b}$ for each pair of algorithms a and b, where $a, b \in \mathcal{A}$ and $\mathcal{A}$ is the algorithm set. The model $f_{a,b}$ predicts the performance difference $d_{i_{a,b}} = f_{a,b}(\mathbf{x}_i)$ between algorithms a and b when applied to the dataset described by the meta-feature vector $\mathbf{x}_i$. For a set of A algorithms, there are $\binom{A}{2} = \frac{A(A-1)}{2}$ such pairwise combinations attempted, leading to a comprehensive assessment of relative algorithm performances.

Once the models are trained, for each dataset i, we record a "win" for algorithm a over b based on the following criteria: If the goal is to maximize a performance metric, a win is recorded if $d_{i_{a,b}} > 0$; conversely, if the goal is to minimize a performance metric, a win is recorded if $d_{i_{a,b}} < 0$. We sum the "wins" for each algorithm across all pairs. The algorithm that accumulates the highest number of "wins" across all comparisons is selected as the optimal choice for that dataset.

5.3.2.2 Multi-output pairwise regression

Alternatively, the multi-output pairwise regression trains a single model f that outputs a performance difference prediction for all pairs of algorithms simultaneously. This method leverages potential correlations among the performance differences to enhance prediction

accuracy. Following the single-output method, we determine the optimal algorithm for each dataset by counting the number of "wins" each algorithm accumulates over all others in the pairwise comparisons, and use a majority vote to select the best-performing algorithm based on the predicted differences.

5.3.3 Classification approach

The classification approach frames the AS problem as a multi-class classification task, where each class uniquely corresponds to one of the algorithms in the algorithm portfolio $\mathcal{A}$. In this approach, a single classifier is trained to determine the optimal algorithm from the set $\mathcal{A}$. Here, the output y_i is a class label corresponding to the name of the optimal algorithm for dataset i characterized by the feature vector $\mathbf{x}_i$. This direct mapping simplifies the process of selecting the most appropriate algorithm for each dataset.

5.3.4 Pairwise classification approach

Pairwise classification transforms the multi-class AS problem into a sequence of binary classification tasks, employing a class binarization strategy. This method decomposes the selection process into multiple two-class problems, where each problem involves making a direct comparison between two algorithms.

5.3.4.1 Single-output pairwise classification

In single-output pairwise classification, the objective is to determine the superior algorithm between each pair from the set $\mathcal{A}$ for a specific dataset. For each pair of algorithms (a, b), a binary classifier $f_{a,b}$ is trained. This classifier processes the meta-feature vector $\mathbf{x}_i$ associated with the $i-th$ dataset to predict which algorithm, a or b, is likely to perform better. The output $d_{i,a,b}$ is a binary label, where the value is 1 if algorithm a is predicted to outperform algorithm b, and 0 otherwise.

Similarly to the pairwise regression approach, this methodology involves training $\binom{A}{2}$ binary classifiers, corresponding to all possible pairs of the A algorithms in the portfolio. The outcomes of these pairwise predictions are then aggregated to determine the most suitable algorithm for each dataset, effectively establishing a "winner" based on the pairwise comparisons.

5.3.4.2 Multi-output pairwise classification

In the multi-output pairwise classification scenario, a single model is utilized to address all pairwise binary classification tasks simultaneously. This model takes a dataset's meta-feature vector $\mathbf{x}_i$ and outputs a tuple $\mathbf{y}_i$ containing $\binom{A}{2}$ binary predictions, each corresponding to a different pair of algorithms. These predictions are then aggregated in the same way as in the single-output approach, where the algorithm accumulating the highest number of favorable outcomes across all pairs is selected as the most optimal for that particular dataset.

5.3.4.3 Cost-sensitive single-output pairwise classification

This approach extends the single-output pairwise classification by introducing a weighting mechanism for the training instances that utilizes the performance differences between each pair of algorithms. The weights are determined by the magnitude of these differences, which is interpreted as the cost associated with the prediction error [174].

The rationale behind this strategy is the prioritization of predictive performance in cases where the performance differences are significant, as these are more impactful on the overall decision-making process. Conversely, a smaller weight is assigned where the performance difference is minimal, thus reducing the penalty for any incorrect predictions in these cases. Therefore, each training instance i comparing algorithms a and b is assigned a weight proportional to the absolute difference in their performances: $|p_{i_a} - p_{i_b}|$, where p_{i_a} and p_{i_b} denote the measured performances of algorithms a and b on dataset i, respectively.

5.4 Experimental Setup

This section details the experimental setup, encompassing the dataset portfolio, associated landscape data, the MLC algorithm portfolio, and corresponding performance data. We also describe the process of training the ML models that underpin the algorithm selector and describe the evaluation of the algorithm selector.

5.4.1 Dataset portfolio and landscape data

The dataset portfolio comprises 40 MLC datasets previously utilized in a benchmarking studies [124]. The datasets come from five different application domains, including text, multimedia, bioinformatics, medical, and chemistry. These datasets exhibit considerable diversity, with label counts ranging from four to 274, data instance numbers from 139 to 17,190, and descriptive features from 33 to 49,060.

For building the algorithm selector, our ML pipeline relies on meta-descriptors (or meta-features) of these MLC datasets that depict the landscape characteristics of the datasets. We utilize the set of 63 MLC meta-features available in our BenchMLC catalogue (see Table 3.4). We refine this initial set by applying Spearman correlation analysis, removing one feature from each pair having a correlation higher than 0.9. All datasets and their corresponding meta-features can be accessed and downloaded through our publicly available BenchMLC catalogue as detailed in Chapter 3.

5.4.2 Algorithm portfolio and performance data

We utilize performance data from execution of 26 MLC algorithms corresponding to our dataset of 40 MLC datasets. This data is available through the BenchMLC catalogue and originates from a comprehensive comparative study of MLC algorithms [124]. This extensive study evaluates the algorithms using a total of 20 performance metrics, which include 18 predictive performance metrics and two efficiency metrics. In this dissertation, we concentrate solely on the predictive aspects of performance, thereby excluding the two efficiency-related metrics from our analysis. Among the 18 predictive performance metrics available, we select five performance metrics that are also used in a related study [44], which examines the data for predicting algorithm performance.

The only prerequisite for applying AS techniques is that there exists (or that there can be constructed) a set of complementary algorithms [175]. Complementary algorithms are those that demonstrate diverse performance across a range of problems, ensuring that the algorithm portfolio effectively addresses varied problem characteristics. Therefore, to assemble a portfolio of MLC algorithms that demonstrates complementary performance across the datasets, we analyze how often each algorithm ranks as the best performer. We then select the top five algorithms for each performance metric to include in our algorithm portfolio. The composition of this final portfolio for each performance metric is detailed in Table 5.1. For more details on the performance metrics and the MLC algorithms, we refer the reader to [124].

Table 5.1: A list of the five performance metrics and the corresponding algorithm portfolios.

Performance Metric	**Algorithm Portfolio**
Hamming loss	DEEP4 [176], RFPCT [33], CC [177],
example-based	AdaBoost.MH [178], TREMLC [179]
F1 macro	CLR [180], [181], AdaBoost.MH [178], RFDTBR [182],
	CC [177], RSMLCC
F1 micro	RFDTBR [182], RFPCT [33], AdaBoost.MH [178],
	CLR [180], [181], PSt [183]
AUROC micro	RFPCT [33], PSt [183], RFDTBR [182],
	EBRJ48 [184], TREMLC [179]
F1 example-based	RFPCT [33], RFDTBR [182], RSLP,
	PSt [183], AdaBoost.MH [178]

5.4.3 Model training and validation

All ML models are built using the Random Forest (RF) algorithm as implemented in the Python package `scikit-learn` [185]. Depending on the ML approach, we employ different variants: an RF classifier for the classification and pairwise classification approaches, and an RF regressor for the regression and pairwise regression approaches. Additionally, the multi-output variant of RF is used for multi-output methodologies.

The ML models are evaluated using a *leave-one-instance-out* strategy, where an instance corresponds to one MLC dataset. Given our portfolio of 40 MLC datasets, this involves training the models 40 times, each time withholding one dataset for testing and using the remaining 39 for training. The evaluation metrics used are mean squared error for regression models and accuracy for classification models, with results averaged across all test instances. We use the default configuration of the RF algorithm without performing hyperparameter tuning.

After training, the raw predictions on the test instances are used to select the best-performing algorithm, as detailed in Section 5.3.

5.4.4 Evaluation of MLC AS

The quality of the AS is assessed by comparing it against two established baselines. The first is the *Virtual Best Solver* (VBS), or "oracle selector", which represents a theoretically perfect selector that always chooses the best performing algorithm for each dataset instance. This baseline serves as an upper limit on the performance of any realistically achievable AS. The second baseline, the *Single Best Solver* (SBS), is the algorithm that exhibits the best average performance across the entire portfolio of datasets, serving as a lower bound for the AS. Note that here, the term "solver" refers to the MLC algorithms being evaluated.

To evaluate the performance of the algorithm selector, we compute the absolute difference between the performance of the selected algorithm A and the best algorithm A^* for a given dataset. The performance, denoted by $p(A)$ for the selected algorithm and $p(A^*)$ for the best algorithm, is measured with respect to a given performance metric. More precisely, this difference is defined as:

$$L(A, A^*) = |p(A) - p(A^*)|$$

This calculation provides a performance measure of the AS for each dataset, allowing us to analyze the distribution of these "losses" across all datasets.

To evaluate the overall performance of our MLC AS, we compute the total loss across all datasets. The total loss is defined as the sum of losses between the algorithm predicted

by the AS to be the best and the VBS for each dataset. This is mathematically expressed as:

$$\text{Total Loss}_{\text{AS}} = \sum_{i=1}^{D} L(A_{\text{AS}_i}, A_{\text{VBS}_i})$$

where A_{AS_i} represents the algorithm predicted by the AS for the i-th dataset, and A_{VBS_i} the VBS for the same dataset and D is the total number of datasets. This total loss metric quantifies the effectiveness of the AS relative to the performance of the VBS across all datasets.

Similarly, the total loss for the SBS is calculated as the sum of losses between the SBS and the VBS across all datasets:

$$\text{Total Loss}_{\text{SBS}} = \sum_{i=1}^{D} L(A_{\text{SBS}_i}, A_{\text{VBS}_i})$$

where A_{SBS_i} represents the algorithm predicted by the SBS for the i-th dataset, and A_{VBS_i} the VBS for the same dataset and D is the total number of datasets.

Note that in the case of pairwise models, it is possible for multiple algorithms to achieve the same number of wins on a given dataset. In these cases, to calculate the $p(A_{\text{AS}_i})$, we use the average value of the measured performance for the set of predicted algorithms.

Finally, the "% of closed VBS-SBS gap" quantifies the percentage reduction in total loss achieved by the AS compared to the SBS. This metric is calculated as:

$$\%\text{ of VBS-SBS gap closed} = \left(1 - \frac{\text{Total Loss}_{\text{AS}}}{\text{Total Loss}_{\text{SBS}}}\right) \times 100\%$$

This measures how much closer the performance of the AS approaches the ideal scenario represented by the VBS, relative to the SBS. A value of 100% indicates that the AS has reached the highest possible performance, matching the performance of the VBS. A value of 0% signifies no improvement over the SBS, while negative values indicate that the AS performs worse than the SBS.

5.5 Results and Discussion

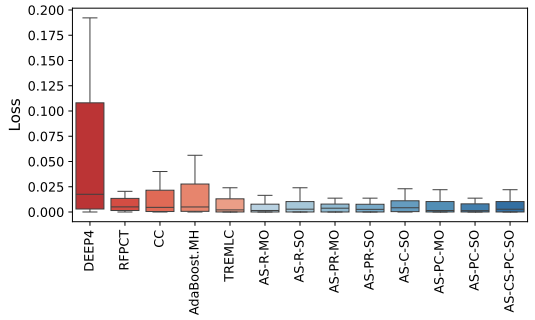
This section presents the experimental results for various ML-based algorithm selectors and includes a discussion on explainability of ML models in the context of AS.

5.5.1 Performance comparison of the different ML approaches for AS

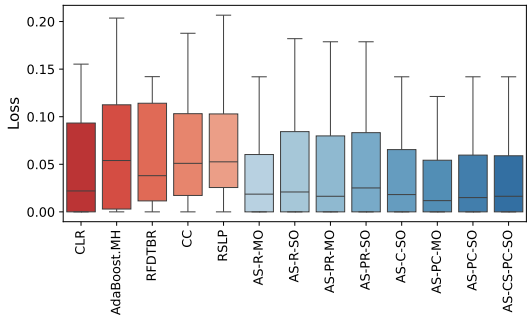
Following the experimental setup described in Section 5.4, we affirm that the AS leads to performance gains compared to any standalone solver (i.e., MLC algorithm).

Figure 5.1 shows the loss (computed as described in Section 5.4.4), across all 40 datasets for the five MLC algorithms when each is statically selected as the best and the eight ML approaches for AS shown on the x-axis, evaluated using the five selected performance metrics. The results show that the loss for the algorithm selectors built using ML techniques is consistently smaller compared to the loss for the five algorithms when each is statically selected as the best algorithm, demonstrating that the ML-based algorithm selectors can adapt to different performance evaluation criteria.

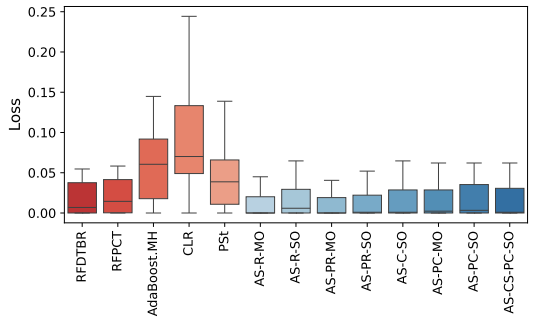
Additionally, the Figure 5.1 illustrates that the results are highly dependent on the performance metric used. This variation is expected because for each performance metric we have a different algorithm portfolio (see Table 5.1).



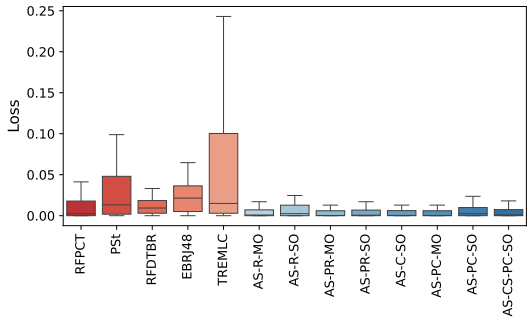
(a) Hamming loss example-based



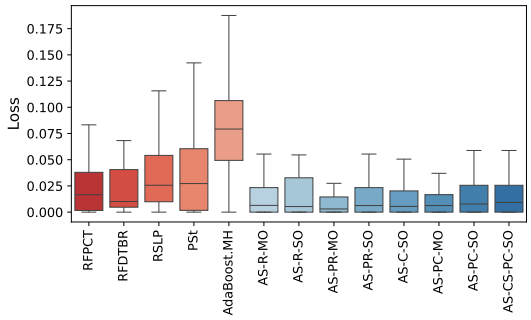
(b) F1 macro



(c) F1 micro



(d) AUROC micro



(e) F1 example-based

Figure 5.1: The loss (measured as the absolute difference of the performance of the VBS and the performance of the predicted best algorithm) of the static selectors (in red) and of the AS (in blue) across the eight different ML approaches (i.e., R-MO = multi-output regression, R-SO = single-output regression, PR-MO = multi-output pairwise regression, PR-SO = single-output pairwise regression, C-SO = single-output classification, PC-MO = multi-output pairwise classification, PC-SO = single-output pairwise classification, CS-PC-SO = cost-sensitive single-output pairwise classification) for evaluation measures a) Hamming loss example-based, b) F1 macro, c) F1 micro, d) AUROC micro, e) F1 example-based.

Figure 5.2: A heatmap depicting the percentage of the VBS-SBS gap closed with the different AS approaches across the five different performance metrics. The final row in the heatmap represents the mean percentage of gap closed across all performance metrics.

We also observe differences in the loss distribution among the AS built with the various ML approaches. This indicates that depending on the experimental setup (i.e., performance metric, algorithm portfolio), different ML approaches may perform best.

To better quantify the performance differences of the various algorithm selectors, we calculate the percentage of the VBS-SBS gap closed (as described in Section 5.4.4) by each ML approach across different performance metrics. Figure 5.2 illustrates these percentages. It is important to note that the SBS is the MLC algorithm that performs the best on the largest number of datasets for a given performance metric. For each performance metric, a different MLC algorithm may be chosen as the SBS based on the number of datasets where it achieves the best performance.

The percentage of the VBS-SBS gap closed, averaged over the performance metrics, ranges between 38.51% and 53.18% across the different ML approaches. The highest percentage of the VBS-SBS gap closed is achieved with the multi-output pairwise regression model at 53.18%, followed closely by the multi-output regression model, which achieves 52.82%. An interesting pattern we observe is that the multi-output models generally perform better compared to their single-output counterparts: 52.82% versus 38.51% for multi-output and single-output regression; 53.18% versus 49.66% for multi-output and single-output pairwise regression; and 47.27% versus 43.45% for multi-output and single-output pairwise classification. This better performance of the multi-output models can be explained by the fact that considering multiple targets can lead to less overfitting to individual targets, and can also exploit the correlations between targets, resulting in more robust and generalized models.

It should be noted that even though, on average (averaged over the five performance metrics), the multi-output models outperform the single-output models, in the case of the Hamming loss example-based performance metric, the opposite is true for the pairwise regression and pairwise classification models. This could be due to the fact that the Hamming loss metric is particularly sensitive to specific types of errors that single-output models handle more effectively in certain scenarios.

Another interesting observation is that the cost-sensitive single-output pairwise classification model does not outperform the single-output pairwise classification model in 3 out of 5 cases and, on average, has very similar performance: 43.45% for the cost-sensitive single-output pairwise classification versus 43.04% for the single-output pairwise classification. Typically, one might expect the cost-sensitive model to exhibit better performance because it specifically emphasizes data instances where the performance discrepancy between two algorithms is significant, thereby prioritizing areas of larger error. However, a possible explanation for the similar performance levels is that the meta-features in our models may not adequately capture the complexity needed to effectively leverage the cost-sensitive adjustments. Moreover, the model's focus on data examples with larger weights might result in overfitting these particular points, which compromises its ability to generalize effectively to new, unseen data.

To intuitively quantify the performance (and thus reliability) of all considered approaches, for the different ML approaches, we measure the ratio at which each MLC algorithm from the portfolio is identified as the VBS compared to how often it is selected by the algorithm selector across different ML approaches. It's important to note that in some instances, especially due to ties in performance or within pairwise models where ties are possible, multiple algorithms may be designated as the VBS or be recommended as the best. Consequently, the sum of ratios in each column might not add up to one. Our observations reveal that all AS approaches generally align with the overall distribution of VBSs, as depicted in Figure 5.3. Across all ML approaches, we note that the AS tends to select more frequently the MLC algorithm that is the best performer in the majority of cases. Conversely, MLC algorithms that are infrequently the VBS are seldom chosen. For example, when assessing MLC algorithm performance using the F1 micro metric, all ML approaches for AS predominantly favor the RFPCT and RFDTBR algorithms, which frequently emerge as VBS, while the CLR algorithm, which seldom is the VBS, is rarely selected.

5.5.2 Discussion on explainable AS

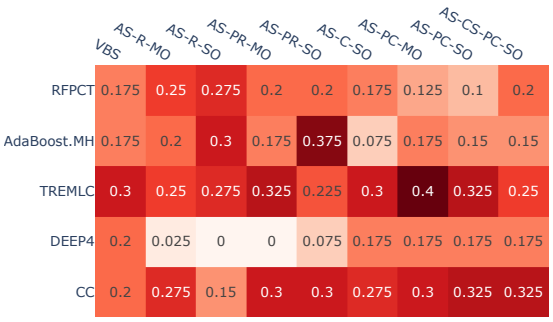
Explainability in algorithm selection (AS) is essential as it ensures that the decision-making process of the ML models that select the best performing algorithm is transparent, comprehensible, and trustworthy. By providing insights into how the models make decisions, we can better understand and effectively use them in various real-world applications.

In this dissertation, all of the ML models for MLC AS that we consider are based on tree ensembles. While tree ensembles are known for their strong performance, they inherently have limited explainability due to their complex structure. To enhance the explainability of these models, we can employ the SHapley Additive exPlanations (SHAP) values technique [11].

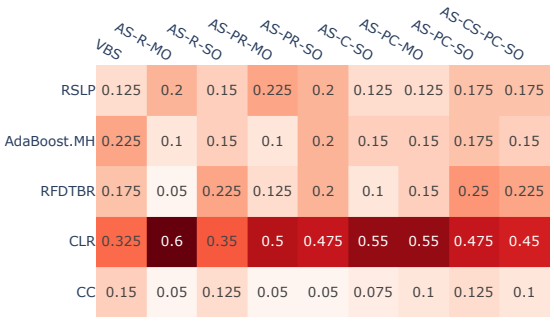
SHAP provides a way to understand the contribution of each feature to the model's predictions, offering a clearer picture of how decisions are made. SHAP can explain the model's predictions on a local level, i.e., explaining individual predictions. By aggregating these local explanations, it is also possible to derive global explanations, offering a comprehensive understanding of the overall model behavior.

When using SHAP for MLC AS, we need to consider the specific ML approach employed for AS. The simplest approach is classification, where we have a single target that directly predicts the best performing algorithm. In this case, SHAP values are calculated using the classification model with respect to this single target. This provides insight into which features are most influential in making the classification decision.

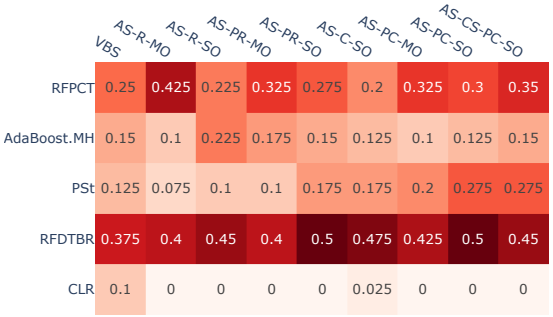
A more complex scenario arises when using regression models for AS. Given an algorithm portfolio of size N, we predict the performance of N output variables, each corre-



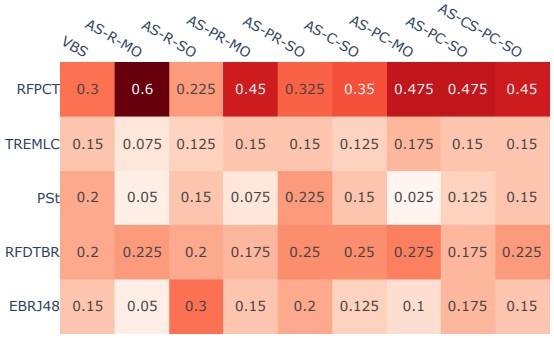
(a) Hamming loss example-based



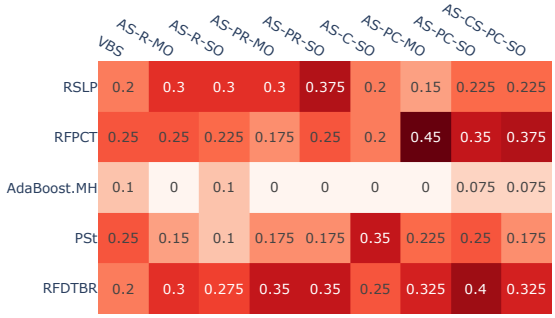
(b) F1 macro



(c) F1 micro



(d) AUROC micro



(e) F1 example-based

Figure 5.3: Heatmaps showing the percentage of datasets where each MLC algorithm is the VBS (column 1) vs. the percentage of datasets where the algorithm is recommended by the AS across the eight different ML approaches (i.e., R-MO = multi-output regression, R-SO = single-output regression, PR-MO = multi-output pairwise regression, PR-SO = single-output pairwise regression, C-SO = single-output classification, PC-MO = multi-output pairwise classification, PC-SO = single-output pairwise classification, CS-PC-SO = cost-sensitive single-output pairwise classification) for evaluation measures a) Hamming loss example-based, b) F1 macro, c) F1 micro, d) AUROC micro, e) F1 example-based.

sponding to an algorithm in the portfolio. These predictions can be made using either single-output or multi-output models. For both types of regression models, SHAP values are calculated separately for each output.

To obtain these SHAP values for the algorithm selector, we first identify the algorithm predicted to perform the best, considering the predictions of all regression models. Then, we retrieve the SHAP values for the specific prediction from the regression model trained for the selected algorithm. This approach clarifies why a particular algorithm was chosen by highlighting the features that significantly contributed to its predicted performance.

Figure 5.4 illustrates the SHAP values for the multi-output regression case for each MLC dataset separately. In this figure, we group the MLC datasets according to the application domains (Text, Bioinformatics, Multimedia, Medical, and Chemistry). We can observe the negative and positive marginal contributions of each MLC meta-feature. Features such as *TotalDistinctClasses, Labels*, *Max IR inter class*, *Max IR intra class*, *Proportion of unique label combination (PUniq)*, and *Ratio of unconditionally dependent label pairs by chi-square test appear to be more important.* Patterns across different application domains are also evident. For example, the *TotalDistinctClasses* and *Labels* meta-features predominantly have negative marginal contributions in the Text domain, while they have positive contributions in the Bioinformatics domain.

Among the ML approaches we consider, pairwise models are also included. Here, we discuss how SHAP values can be applied in this scenario.

When using pairwise models for AS, calculating SHAP values becomes more complex due to the nature of the comparisons. Pairwise models compare pairs of algorithms to determine which one performs better. To calculate SHAP values for the selected/predicted algorithm, we must consider each pairwise model where the selected/predicted algorithm is one of the two being compared, and the model predicted it as the better performing algorithm.

Since each pairwise model has a different target (the relative performance between two specific algorithms), direct aggregation of SHAP values across these different models is not feasible. This complexity reduces the interpretability of the models, making it challenging to combine insights from multiple targets.

One approach to address this is to rank the features based on their absolute SHAP values for each pairwise model. By doing so, we would be able to identify the most important features across all comparisons. After ranking the features for each pairwise model, we can calculate the average ranks to get an overall sense of feature importance. However, this method loses the information about whether a feature's contribution is positive or negative, as it only considers the magnitude of importance.

Another approach is to examine each individual pairwise model separately. This retains the information about positive and negative contributions, providing a detailed understanding of feature importance in the context of each specific pairwise comparison. Although this method preserves more detailed information, it requires a more nuanced analysis to interpret the results across multiple pairwise models.

5.6 Summary

In this chapter, we have investigated the potential of automated algorithm selection for the MLC learning task. We have compared various ML approaches, including regression, classification, pairwise regression, pairwise classification, and cost-sensitive pairwise classification models, to determine their effectiveness in predicting the best-performing algorithms for diverse MLC datasets. Additionally, we examined the difference between single- and multi-output models for the AS ML approaches where this is applicable.

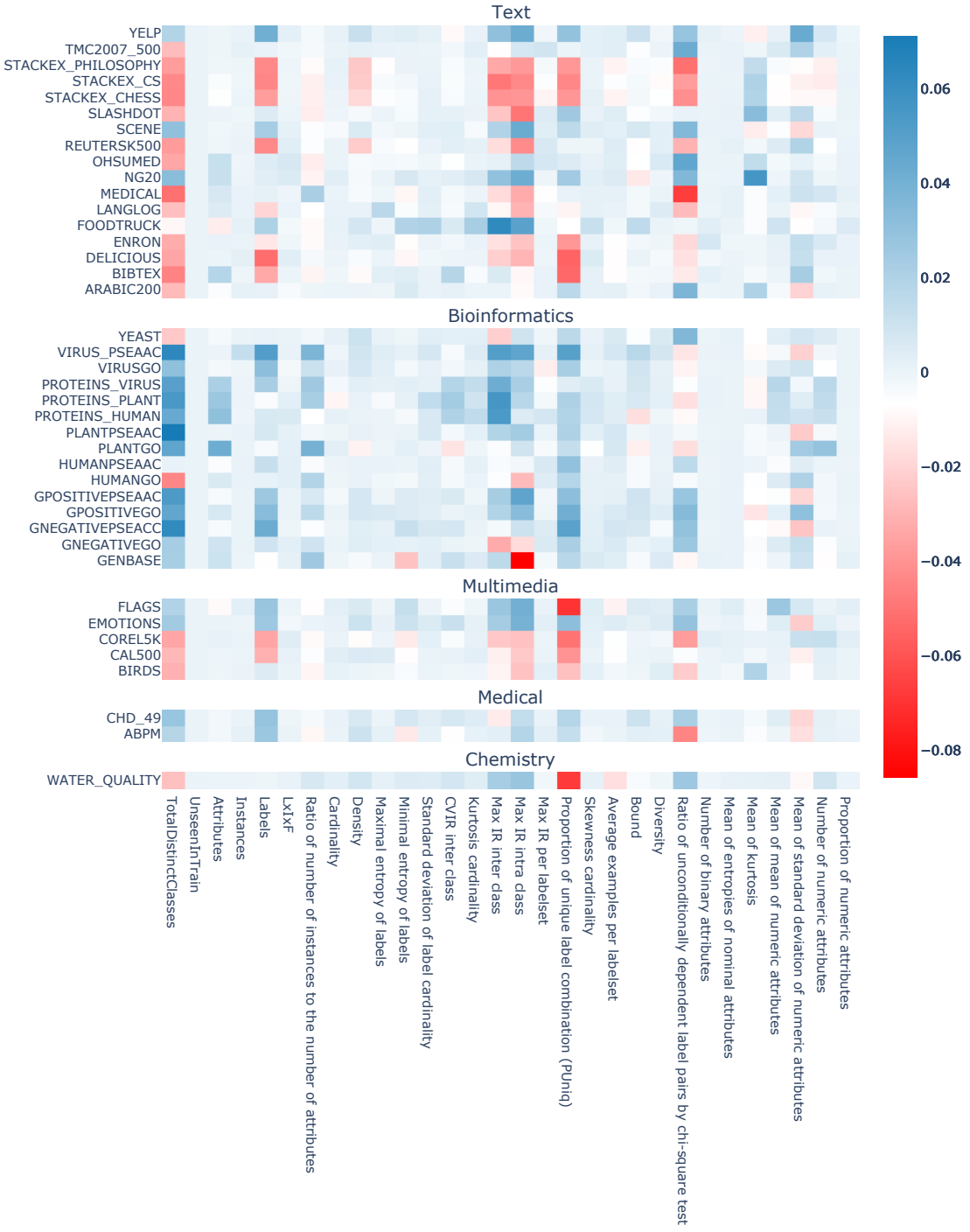


Figure 5.4: SHAP feature importance scores for MLC AS based on multi-output regression. The feature importances for each MLC dataset are grouped by application domain and obtained on the test set using leave-instance-out validation.

We evaluated five performance metrics across 40 MLC datasets. The results demonstrated that all ML approaches for AS yielded performance gains over the scenario where a single MLC algorithm is used (the one that, on average, performs best across all datasets), regardless of the evaluation measure. However, the results highly depend on the MLC performance metric.

Notably, multi-output models generally outperformed their single-output counterparts. This suggests that considering multiple targets simultaneously can enhance the robustness and generalization of the AS models by leveraging correlations between targets.

Despite the overall superior performance of multi-output models, some performance metrics, such as Hamming loss example-based, showed that single-output models may be more effective in specific scenarios. This indicates that the choice of ML approach may need to be tailored to the specific performance metric.

The cost-sensitive single-output pairwise classification model did not significantly outperform the standard single-output pairwise classification model. This suggests that the meta-features used may not adequately capture the complexity required to leverage cost-sensitive adjustments effectively.

The discussion about SHAP feature importance highlighted the varying complexity of calculating SHAP scores across different ML approaches. Classification is the simplest, followed by regression, while pairwise models are the most complex due to the different pairwise targets, making explainability significantly more challenging.

Chapter 6

Using Machine Learning Methods to Assess Module Performance Contribution in Modular Optimization Frameworks

In this chapter, we show how benchmarking data from the OPTION KB can be exploited to assess modular optimization algorithms. We propose a data-driven methodology to evaluate the performance contribution of individual modules within two modular optimization frameworks: modCMA-ES [78] and modDE [79]. By training algorithm performance models and using Shapley values to explain their predictions, the methodology links the landscape characteristics of BBOB problems with the performance of various algorithm variants (created by combining different modules). This approach provides insights into which landscape characteristics most significantly impact performance predictions.

The chapter is structured as follows. In Section 6.2, we review related work on empirical performance analysis of modular optimization algorithms, automated algorithm performance prediction, and explainable ML. Section 6.3 presents our methodology for obtaining algorithm meta-representations and using them to predict the algorithm's modular configuration. We describe our experimental design in Section 6.4. In Section 6.5, we discuss the key findings and results of our experiments. Finally, in Section 6.6, we summarize our contributions and outline several directions for future work.

This chapter is based on the article "Using Machine Learning Methods to Assess Module Performance Contribution in Modular Optimization Frameworks " [186], published in Evolutionary Computation Journal, MIT Press, 2024 and on the paper "The importance of landscape features for performance prediction of modular CMA-ES variants" [187], which appeared in the Proceedings of the the Genetic and Evolutionary Computation Conference (GECCO), 2022.

All data and code related to this chapter are publicly available on GitHub at: `https://github.com/KostovskaAna/AssessingModuleContribution`.

6.1 Problem Definition

Many state-of-the-art black-box optimization algorithms are claimed to have been originally inspired by natural processes such as evolution and swarm intelligence [188]. Driven by the varying performance of algorithms across different problem types, researchers continue to seek inspiration from nature and employ diverse metaphors to develop and refine

these techniques. However, a recent call for action by the evolutionary computation scientific community has highlighted three major concerns about metaphor-based metaheuristics [189]. Firstly, the usefulness of metaphors in metaheuristics is questionable, as many "novel" algorithms inspired by metaphors often lack scientific justification and oversimplify or modify the metaphor to resemble an optimization process, making them differ greatly from their original inspiration. Secondly, there is a lack of originality, with researchers often rediscovering concepts that have already been published in earlier studies (under different names). Finally, the experimental validation and comparisons of these algorithms are often biased, with improper comparisons made between novel and non-state-of-the-art algorithms on benchmark problem instances that are under-representative of the diversity in the problem space.

These issues underscore the need for novel approaches to better understand the behavior of metaphor-based metaheuristics (and metaheuristics in general) in order to identify genuine contributions to the field.

A commonly used method to understand the behavior of algorithms is the assessment of their performance through benchmarking and statistical analyses. Typically, this involves reporting the average performance across a selected set of benchmark problems [57], [190]. However, this approach has faced criticism for its limitations in accurately interpreting algorithm behavior and its inability to generalize to new problems [191]–[193]. Furthermore, in these statistical analysis approaches, algorithms are treated as black-boxes much like optimization problems, hence, it is challenging to draw any conclusions about the characteristics of the algorithms that contribute most to their performance.

Another approach to understanding metaheuristics is the development of classification systems and taxonomies that try to categorize these algorithms based on their underlying mechanisms, search strategies, and other relevant factors [159], [194], [195]. Unlike statistical approaches that treat algorithms as black boxes, these classification systems aim to provide a structured way of describing metaheuristics and help researchers identify similarities and differences between different algorithms. However, one limitation of these classification systems is the lack of connection between the algorithms and the optimization problems they are designed to solve, as well as the performance they exhibit on these optimization problems. Without this connection, it can be challenging to understand the performance of the algorithms on specific problem instances.

To overcome these limitations, new methods for assessing algorithm behavior are needed. These methods should consider the characteristics of the algorithm, the landscape characteristics of the problems, and the interaction between the two in terms of their influence on performance behavior. By understanding these factors, we can develop more effective algorithms that perform well on a range of problem instances.

One promising approach for improving the assessment of algorithm behavior is to use modular optimization algorithm frameworks [78], [81], [83], [196], which we have introduced in Chapter 2. These frameworks provide a flexible and modular way to design and evaluate metaheuristic algorithms. In these frameworks, 'modules' essentially represent the operators in optimization algorithms. For clarity and consistency throughout this dissertation, we use the term 'module' instead of 'operator' in the context of modular frameworks. Modular optimization algorithm frameworks can also provide a way to bridge the gap between algorithm behavior and the optimization problems they are designed to solve. By designing algorithms as collections of interchangeable components, researchers can test different combinations of components on a variety of problem instances.

In this dissertation, we use modular optimization algorithm frameworks to assess the different algorithmic ideas that were proposed in the literature. Our analysis is focused on examining each module individually, reflecting the common practice of proposing al-

gorithmic ideas in isolation. Exploring the interactions between these modules and their collective impact on performance is an interesting aspect recently examined in [197], but it remains outside the scope of this dissertation.

We focus on the analysis Differential Evolution (DE) [76] and Covariance Matrix Adaptation Evolution Strategies (CMA-ES) [77]. We use their decomposed versions on basic components/modules that are available in the modCMA-ES [78] and modDE [79] modular frameworks, respectively.

In this chapter, our main goal is to show development of an empirical workflow for understanding the impact of modules on the performance of DE and CMA-ES algorithm. We will analyze 324 modCMA-ES and 576 modDE algorithm variants across 24 BBOB problems to: (i) Evaluate the effect of individual modules on overall algorithm performance through analysis of performance data; (ii) Train ML regression models to predict algorithm performance, with a focus on understanding how problem landscape features impact these predictions. This approach provides an explainable ML model, linking feature importance directly to the model outcomes; and (iii) Train classifiers that use performance and landscape feature importance data to predict algorithm module configurations. Obtaining high prediction accuracy signals variability in the performance data w.r.t. to the modules, suggesting that higher accuracy reflects greater variability or a stronger impact of the module on performance. Conversely, having lower accuracy suggests a module's configuration has a minimal impact on overall performance.

6.2 Related Work

Diverse research has investigated the modular CMA-ES and DE algorithm families in various single-objective learning scenarios. This includes conducting empirical performance analysis of CMA-ES [198] and DE [199], predicting CMA-ES [200] and DE [201] algorithm performance, automated algorithm selection [202], and automated algorithm configuration [67], [203].

The empirical performance analysis [198], [199] has focused on providing empirical results through descriptive statistics of the performance achieved on a particular benchmark suite. Another way to compare algorithms' behavior using information from the performance space is to use *performance2vec* meta-representations [204]. Here, the results obtained by multiple runs of an algorithm instance on a particular problem are averaged and stored as a vector representation that consists of the results for all benchmark problems. Further, the similarity between algorithm instances is assessed as the similarity between the vector representations obtained by using *performance2vec*.

The studies performed in automated algorithm performance prediction allow us to develop an explainable ML predictive model. For this purpose, landscape properties [205] of the problem instances are used as input features to train an ML predictive model that links them to the performance of the algorithm achieved after some function evaluations. Further, by applying post-hoc explainable techniques, the contribution of each landscape feature to the accuracy of performance prediction can be analyzed. Recently, the SHAP [206] feature ranking method has been explored for such analyses, since it provides explanations both at a *global* level (i.e., all benchmark problem instances) and at a *local* level (i.e., per problem instance). The SHAP explanations can be used for algorithm behavior meta-representation that facilitates the capture of the interactions between the problem landscape properties and the performance of the algorithm instance. These meta-representations have been used with unsupervised techniques to find similar groups of algorithm behavior of CMA-ES [200] and DE [201] configurations.

The mentioned studies integrate into a broader range of research that aims to under-

stand the behavior of modular CMA-ES and modular DE. However, despite significant efforts in this direction, most of the studies that focus on automated algorithm performance prediction and selection treat the CMA-ES or DE configurations as black boxes, without exploring the impact of the individual modules on the final performance of a configuration. While some studies have used time-series features calculated from the global state variables to classify isolated CMA-ES modules [207], there is no information on how these features are linked to each module separately. Another study [203] has investigated a problem instance-based configuration model that selects optimal CMA-ES modules using landscape features of problem instances but does not provide any insight into the importance of the landscape features.

Modular algorithm components have also been investigated in multi-objective optimization. [208] focus on the automatic design of novel multi-objective evolutionary algorithms (MOEAs) through the utilization of a conceptual framework encompassing various MOEA components. However, the study does not investigate the impact of each of those modules on the overall performance of the algorithm, nor does it provide insight into the importance of problem landscape features.

A purely performance-oriented view on the modular algorithm framework was taken by [209], where a modular suite of pseudo-Boolean optimization algorithms is implemented within the ParadisEO framework [80] and tuned on a collection of W-model problem instances [210], [211] using the irace algorithm configurator [212]. Here, the goal is to identify module combinations that work well together, rather than to explore their complementarity.

6.3 Methodology

Our methodology comprises three main components: (i) generating meta-representations of the modular algorithms (described in Section 6.3.1); (ii) exploratory analysis, investigating the impact of the modules on the performance and investigating the importance of the landscape features for algorithm performance prediction (Section 6.3.2); and (iii) using the learned meta-representations in a supervised classification task to predict the modular configuration of different algorithm instances (Section 6.3.3).

6.3.1 Generating meta-representations of modular algorithms

Algorithm meta-representations are structured representations designed to encapsulate the key characteristics of algorithms, making them well-suited for downstream analysis and predictive tasks. In this chapter, we focus on meta-representations expressed as vector representations, where each algorithm instance is mapped to a fixed-length numerical vector. These vectors encode varying types of information—such as an algorithm's behavior, configuration, and interaction with problem instances—depending on the specific type of meta-representation. This approach provides a standardized framework for comparison and analysis across different algorithm instances. We investigate two types of algorithm meta-representations, *performance-based* and *Shapley-based*.

6.3.1.1 Performance-based meta-representations

Performance-based meta-representations [204] rely solely on performance data, enabling us to develop an understanding of how the different modules contribute to the performance of the algorithm variant. To obtain this data, we execute each modular algorithm variant (i.e., algorithm instance) on a range of problem instances from diverse classes.

Considering the stochastic nature of the algorithms, to obtain reliable estimates of the performance of each variant on each problem instance, we conduct r independent runs of the algorithm variant. In each run, we measure the *precision*, i.e., the absolute difference $f(x^{\text{best}}) - f^*$, between the best solution x^{best} found by the algorithm in the considered run and the global optimum $f^* := \inf_x f(x)$. The solution quality (or performance) for instance j in class i, referred to as q_{ij}, is determined as the median of these precision values.

To summarize the algorithm variant's performance on a problem class level, we calculate the mean $p_i = \frac{1}{m}\sum_{j=1}^{m} q_{ij}$ of the solution qualities q_{ij} across the m instances in class i.

Finally, the overall performance of an algorithm instance across n classes is summarized in an n-dimensional vector $P = (p_1, p_2, ..., p_n)$.

6.3.1.2 Shapley-based meta-representations

Shapley-based meta-representations consist of problem landscape feature importance scores derived from regression models that predict algorithm performance. Details on the Shapley scores are provided in Section 5.5.2.

To construct these meta-representations, we first train regression models for performance prediction for each variant of the modular algorithms, separately. We consider a portfolio of problem classes with size n and m instances of each problem class, resulting in a total of $n \times m$ problem instances. Each problem instance is represented as a vector of ℓ problem landscape features, $(x_1, x_2, ..., x_\ell)$, which serve as input for training the regression models. The target output y that we aim to predict is the algorithm's performance within a fixed budget of function evaluations, as detailed in Section 6.3.1.1.

After training the regression model for performance prediction, we calculate the Shapley values of the landscape features. Applying the Shapley value calculation on the regression models that predict the performance of each algorithm instance separately gives us the Shapley-based meta-representations as an ℓ-dimensional vector, $(s_1, s_2, ..., s_\ell)$. We need to point out here that the Shapley meta-representations are model-specific and depend on the ML algorithm used for learning the predictive model.

6.3.2 Exploratory analysis using the meta-representations

We use the learned algorithm meta-representations in two types of exploratory analysis: (i) to investigate the impact of a module's configuration on the algorithm's performance and (ii) to investigate the importance of the landscape features when predicting the algorithm's performance across the different module configurations.

6.3.2.1 The impact of module configuration on algorithm performance

Here, we make use of the performance-based meta-representations discussed in 6.3.1.1. First, to investigate the impact of module configuration on algorithm performance, from a selected set of algorithm modules and their configurations, we generate all possible configurations of the algorithm variants that we will further investigate. Then, the different algorithm variants are grouped with respect to a given module to observe whether there are some differences in the performance when we change the module configuration and introduce some specific structural changes to the algorithm.

Consider as an example the modular CMA-ES algorithm [78]. This algorithm has multiple configurable modules, but in this illustrative example, for simplicity, we focus on three: elitism (which can take values of either true or false), the base sampler (which offers different sampling techniques such as Gaussian, Sobol', and Halton), and the step size adaptation mechanism (which includes CSA and Step Size Adaptation with PSR).

Table 6.1: An illustrative example of groups of CMA-ES algorithm variants when we investigate the impact of the elitism module on the algorithm's performance.

	Elitism	Base sampler	Step-size adaptation
1	True	Gaussian	CSA
2	True	Gaussian	PSR
3	True	Sobol'	CSA
4	True	Sobol'	PSR
5	True	Halton	CSA
6	True	Halton	PSR

(a) Algorithm variants with elitism

	Elitism	Base sampler	Step-size adaptation
7	False	Gaussian	CSA
8	False	Gaussian	PSR
9	False	Sobol'	CSA
10	False	Sobol'	PSR
11	False	Halton	CSA
12	False	Halton	PSR

(b) Algorithm variants without elitism

By combining the settings of these three modules, we can create a total of 12 different algorithm variants as shown in Table 6.1. To assess the impact of elitism, we divide the configurations into two distinct groups: one with elitism activated (elitism = True) and another without it (elitism = False). This division allows us to analyze and compare the performance of the algorithm variants under different settings, e.g., for elitism.

For each group, we visualize the distribution of the achieved performance on all problem instances and problem classes. Differences in these distributions would indicate that certain modular configurations perform better/worse on the overall problem instance portfolio. We repeat this process for the remaining modules. Additionally, the same analysis can be performed at problem class level to investigate whether there are differences in performance in the different problem classes.

6.3.2.2 The importance of the landscape features in algorithm performance prediction for the different module configurations

Compared to the performance-based meta-representations, the Shapley-based ones (discussed in 6.3.1.2) come with the benefit that they can be employed in an exploratory analysis pipeline where we can investigate the importance of the landscape features across the different modules and across the different configurations of a given module. To this end, we perform the same process of grouping the algorithm variants as described in Section 6.3.2.1. We then calculate the importance of the landscape features for each group separately and average them across all problem instances. We repeat this process for the remaining modules. This approach facilitates the exploratory analysis of the effect each of the modules has on the final performance of the algorithm. Furthermore, trends in the landscape space can be observed. More specifically, some problem landscape features can be found to hold more predictive value than the rest by observing the SHAP values across the different problems (in multiple dimensionalities) for different budgets.

6.3.3 Prediction of a module's configuration of the algorithm instances

The meta-representations (both performance- and Shapley-based) of the modular algorithms variants can be assigned labels that indicate their modular configuration. This labeled data serves as input to train ML classifiers, which predict the configuration of the algorithm's modules. These classifiers are beneficial, for example, in cases where we have the performance data of an algorithm that is achieved after some function evaluations on a particular benchmark suite, but we don't have information about the configuration of the algorithm. By using its meta-representation we may be able to identify a modular

configuration with similar performance behavior. This could help us in a lot of studies for which the performance data is publicly available, but details about the tested configurations are missing. For example, if we have CMA-ES performance data, by using the learned classifiers we can identify a modular CMA-ES configuration with similar behavior.

To test the power of the classifiers, we use the meta-representations of each modular configuration, and we use the classifiers to predict the modules that are activated with their values. Further, we report the F1 score (macro F1 score in the case of multi-class classification) of the predictions across all modules, problem dimensions, and different function evaluation budgets. However, the classifiers may make wrong predictions for the configuration, and the prediction may differ from the true configuration in one or several modules. The wrong predictions affect the performance of the classifier, but the predicted configuration and the true one may still have similar behavior.

To evaluate this, we perform a statistical analysis based on hypothesis testing including the raw performance data for the true and the predicted modular configuration. For this purpose, we use the Deep Statistical Comparison (DSC) approach [190] that ranks the true and the predicted configuration for each problem instance separately, by comparing the distribution of their raw performance data (for each problem instance separately). The ranked data obtained for the true and the predicted configuration across all benchmark problem instances is further analyzed by the Wilcoxon signed-ranks test [213] to find if there is a statistically significant difference in the performance of the true and predicted configuration on the selected benchmark suite.

6.4 Experimental Design

In this section, we provide details on the experimental setup, which consists of several components. We describe our problem portfolio, problem landscape data, algorithm portfolio, and algorithm performance data. Additionally, we provide information on the regression models for algorithm performance prediction and the classifiers for the prediction of the modular configuration of each algorithm instance.

6.4.1 Problem instance portfolio and landscape features

The problem instance portfolio consists of the 24 single-objective, noiseless black-box optimization problems sourced from the BBOB benchmark suite [54] of the COCO benchmark environment [57]. More specifically, we consider the first five instances of each of the 24 BBOB functions, both with dimension $D = 5$ and $D = 30$. This results in two separate problem instance portfolios, one for each dimension, with each portfolio containing a total of 120 problem instances.

To represent the landscape of the problem instances, we utilize 46 "cheap" Exploratory Landscape Analysis (ELA) features implemented in the R package `flacco`[214]. These ELA features are readily available via the OPTION KB (for more details, see Section 4.3.5.4).

In the OPTION KB, ELA features calculated using different sampling techniques and sample sizes are available. For this part of the dissertation, we use ELA features derived from the Sobol' sampling strategy with a sample size of $100D$ across 100 independent repetitions. To represent the landscape of each problem instance, we use the median value for each feature over these 100 repetitions.

We deliberately chose a substantial sample size for ELA computation to mitigate the effects of noisy feature evaluations. Although feature selection has been shown to improve results in performance prediction tasks [215], we did not perform feature selection. We expect our findings to be robust for other types of features as well, based on previous

work [89], [216].

6.4.2 Algorithm portfolio and performance data

We examine two black-box optimization algorithms that have modular implementations available, namely CMA-ES and DE. For CMA-ES, we utilize the modCMA-ES framework [78], which encompasses various versions of the core algorithm. These modifications include changes in the sampling distribution (such as mirrored or orthogonal sampling), weighting schemes for recombination, and restart strategies, to name a few. This modular structure allows for the creation of at least 36 288 configurations of CMA-ES, and additionally provides access to a large set of control parameters (population size, update rates, etc.).

We utilized the modDE [79] package for DE. This package provides a diverse array of mutation mechanisms and modules for selecting the base component, the number of differences included, and the use of an archive for some of the difference components. Additionally, the package enables the usual crossover mechanisms and incorporates update mechanisms for internal parameters based on several state-of-the-art DE versions. In total, this package allows for the creation of at least 1 474 560 configurations of DE.

In this dissertation, we undertake a comprehensive analysis of performance data encompassing 324 algorithm variants for modular CMA-ES and 576 variants for modular DE across the 120 problem instances in 5 and 30 dimensionality. Our objective function measures the precision of the algorithm's solution, i.e., the distance to the optimum, within a fixed budget of function evaluations. We considered six different budget values, $B \in \{50D, 100D, 300D, 500D, 1\,000D, 1\,500D\}$, where D is the problem dimensionality.

For further insights into the modules under examination, as well as the parameter spaces utilized for CMA-ES and DE, alongside a detailed description of the process of generating the performance data, readers are directed to Section 4.3.5.5. All performance data, as well as the detailed representations of the algorithm variants are available in the OPTION KB.

6.4.3 Regression models for algorithm performance prediction

In this study, we train regression models for algorithm performance prediction as part of the pipeline of obtaining Shapley-based algorithm meta-representations. Previous studies have investigated the use of ML in algorithm performance prediction, including the use of Random Forest (RF) regression models [217]–[220]. RF, an ensemble-based decision tree method, is thoroughly described in the seminal work by [39]. In our work, we employ the RF approach to learn performance prediction regression models, as they have been shown to provide promising results in this context [200], [221] and we tune their hyperparameters. For training the models we use the RF algorithm as implemented in the Python package `scikit-learn` [185].

To ensure optimal results, we trained separate regression models (single-output models) for each modular variant. This decision was based on findings by [200], which showed that multi-output models (models that predict the output for several algorithm instances simultaneously) did not demonstrate performance gains compared to single-output models.

For learning the performance prediction models, a vector of 46 ELA features is used to describe each problem instance. Our objective is to predict the precision, i.e., the distance to the optimum that each algorithm in the portfolio will attain on a problem instance, given a fixed budget of function evaluations and problem dimensionality. In this study, we log10-transform the target variable (the median of the 10 independent runs) as it has been shown to improve the performance of the learned predictive models when the target

Table 6.2: Parameters of the RF approach and their corresponding values considered in the grid search.

Hyperparameter	Search space
n_estimators	$[10, 50, 100, 500,]$
max_features	$['auto', 'sqrt', 'log2']$
max_depth	$[4, 8, 15, None]$
min_samples_split	$[2, 5, 10]$

variable is the distance to the optimum [202]. We also cap the target variable to 10^{-8} prior to performing the logarithmic transformation.

Hyperparameter tuning and model evaluation. To assess the learned ML models' performance, we use a nested cross-validation (CV) technique that involves two stages. In the outer loop, we partition the data into training and testing sets, while the inner loop determines the optimal parameters of the ML method. This evaluation approach may require significant computational resources, but yields more reliable estimates of the model's generalization ability as compared to traditional train/val/test data splitting or standard CV [222], [223].

To implement the outer loop, we apply a leave-one-group-out CV, which segments the data into groups/folds based on the unique ID of each problem instance. Since our study involves the first 5 instances of each of the 24 BBOB problems, we create 5 folds by assigning 4 for training and 1 for testing. We repeat this process five times, each time selecting a different fold for testing while using the remaining four for training.

The inner loop adopts a grid search approach to tune the parameters and selects the optimal ones based on the average performance of the inner CV's holdout folds. A leave-one-group-out CV is applied to the training data (i.e., the four folds) obtained from the outer loop. The R^2 score is used as a performance metric. The parameters chosen for tuning and their corresponding search spaces can be found in Table 6.2.

After the optimal parameters have been determined, the model is trained on the entire training data, and its performance is assessed using the test set from the outer loop.

6.4.4 Classification models for predicting/identifying the modular configuration of algorithm variants

To train classifiers that predict the modular configuration of an algorithm variant, we use the algorithm meta-representation as input data and apply the RF classifiers implemented in the Python package `scikit-learn` [185]. We consider two scenarios: (1) Single-output classifiers – we train a classifier for each module separately. Depending on the number of possible configurations for each module, we perform binary classification (when there are 2 possible configurations of the given module, leading to a binary output/target variable) or multi-class classification (when there are more than 2 possible configurations of the given module, resulting in a discrete datatype for the output/target variable), and (2) Multi-output classifiers – we train a single classifier to predict the configuration of all modules simultaneously. Here, the output/target variable is a record of discrete values.

We evaluate both types of classifiers as they have not been studied in this context before. Note that for different problem dimensionalities and function evaluation budgets, we train separate classifiers.

In addition, we assess the performance of TabPFN, a pre-trained Transformer model

that approximates probabilistic inference for a novel prior in a single forward pass. TabPFN was shown to have fast training time and competitive performance on tabular prediction tasks by [224] and we use their implementation.

All classifiers are trained using default (hyper-)parameter values. To evaluate the performance of the learned models, we partition the data into training and testing sets using leave-one-group-out cross-validation, which segments the data into train/test folds based on the unique ID of each benchmark problem instance. As a performance indicator, we report the F1 scores of the classifiers.

6.5 Results and Discussion

Following the methodology described in Section 6.3 and the experimental protocol given in Section 6.4, we first perform an exploratory analysis using the algorithm meta-representations (Section 6.5.1). We then present results on the task of predicting the modular configuration of the algorithm variants from algorithm behavior meta-representations in Section 6.5.2.

6.5.1 Exploratory analysis

6.5.1.1 The impact of the modules on the performance of the algorithms

We investigated how different configurations of modules impact the performance of the CMA-ES and DE algorithms, using performance-based meta-representations in a log-10 scale. The distribution of the precision achieved by different variants of the CMA-ES algorithm on $5D$ problem instances is presented in Figure 6.1. We tested six different modules (elitist, mirrored, base sampler, weights option, local restart, and step size adaptation) across the six function evaluation budgets. Each violin plot in the figure shows the precision across all CMA-ES algorithm variants that have the same value for a given module.

For instance, there were 324 algorithm variants selected as the Cartesian product of the six modules, and 162 algorithm variants had the elitism module activated, while 162 did not. Therefore, the violin plot for "elitism = true" is based on the precision values of 162 algorithm variants, where the precision value of an algorithm variant is the mean value of the performance-based meta-representations (i.e., the mean value of a numerical vector representation of size 24) in a log-10 scale as detailed in Section 3. The precision values are inversely proportional to algorithm performance, with smaller values indicating better performance.

We analyzed the results displayed in Figure 6.1 and made the following observations:

- The activation of elitism in the algorithms leads to improved performance for smaller evaluation budgets. As the budget increases, this trend reverses and elitist configurations are overtaken by their non-elitist counterparts;

- Algorithm variants that have activated mirrored orthogonal sampling with pairwise selection (mirrored pairwise) demonstrate a longer tail towards poorer performance than those that use mirrored sampling without pairwise selection and those that do not use mirrored sampling at all, although on average they perform similarly;

- At the lower budget cut-offs, the Halton sampling showed the best performance, Sobol' sampling came second, and Gaussian sampling demonstrated the worst performance out of the three. As the budget increases, the differences are less evident;

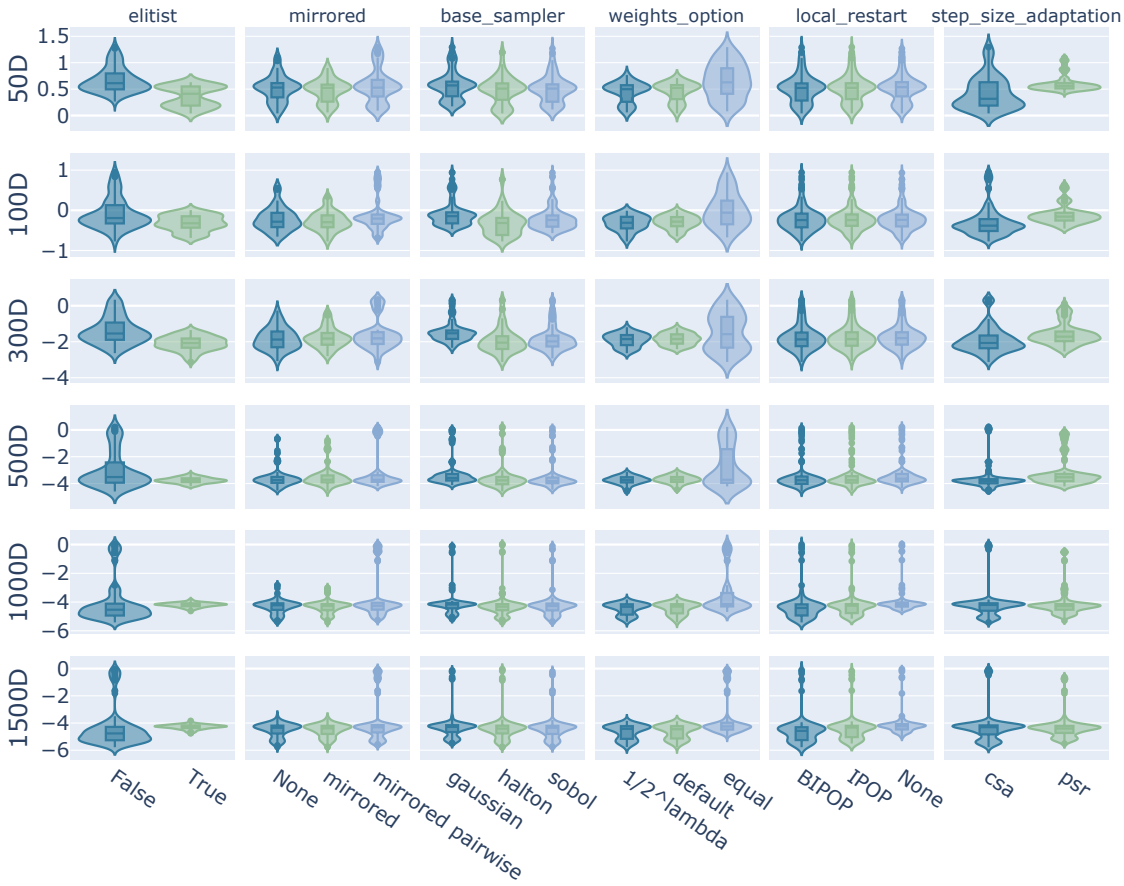


Figure 6.1: Distribution of the precision achieved by different variants of the **CMA-ES** algorithm on $5D$ problem instances for different modular configurations, across different function evaluation budgets. The precision values are inversely proportional to algorithm performance, with smaller values indicating better performance.

- Algorithms with recombination weights set to $(1/2)\hat{}\lambda$ and default weights have similar distributions. Also, all three configuration setups have similar average performance;
- For the lower budgets, we observe that the local restart module achieves comparable performance for the three modular configurations (BIPOP, IPOP, and no restart) across the different budgets. This makes intuitive sense, as at low budgets the algorithm will not have had a chance to trigger any of the restart criteria. As the budget increases, IPOP and BIPOP local restart techniques show slightly better performance compared to algorithm variants without a restart mechanism, which matches observations made in previous work [225]; and
- In the case of step size adaptation, for smaller budgets, cumulative step size adaptation (CSA) exhibits better performance than step size adaptation using the population success rule (PSR).

For the DE configurations, we show the same type of visualization in Figure 6.2. From this figure, we can see that the overall performance differences between DE module options are much smaller than those seen for CMA-ES. The clear exception is the LSPR module that, if enabled, results in much worse performance for smaller budgets. This matches

our intuition since LPSR changes the initial population size to $20D$ at the beginning of the search. This much larger initial population size leads to a slower convergence at the beginning of the search. The difference to no-LPSR slowly decreases over time, but it does not manage to overtake it within our maximum budget of $1\,500D$ function evaluations. This observation also seems to suggest that the population size is a critical parameter of DE, which matches previous observations [226]. For the other modules, we observe that the mutation base and reference settings, which are more elitist (best and pbest) show improved performance for low budgets, matching the observations for CMA-ES.

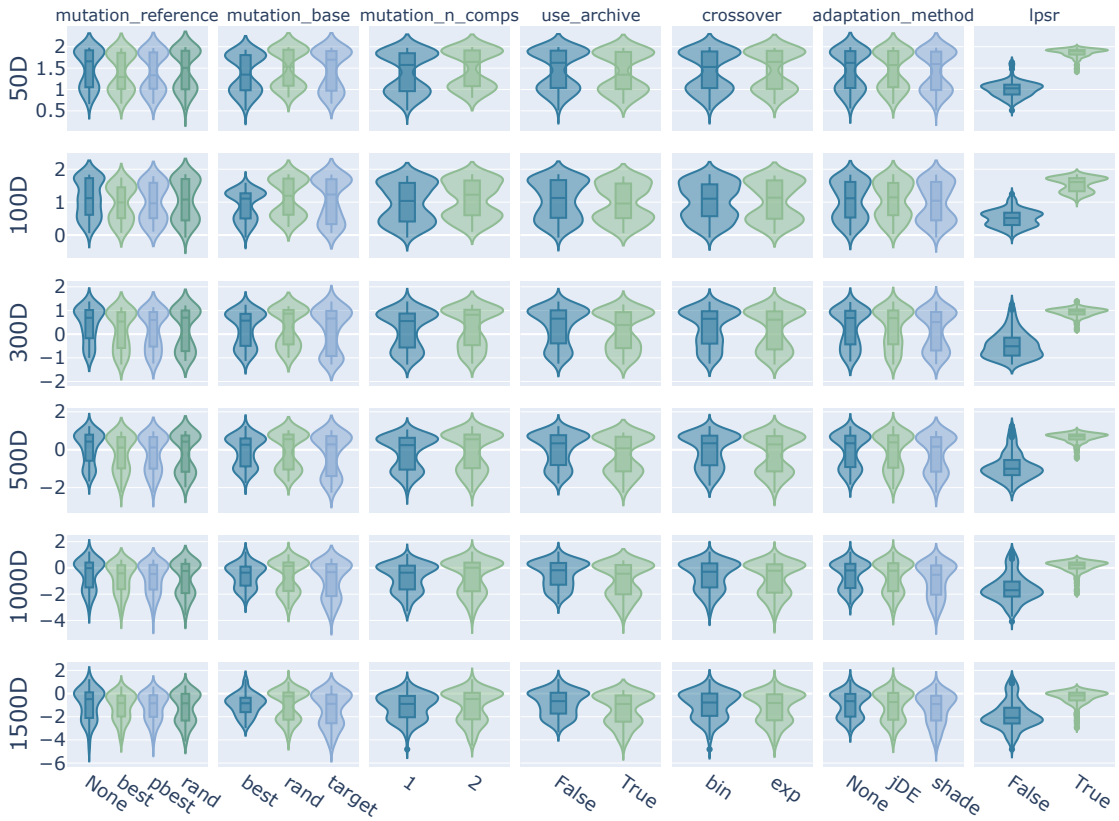


Figure 6.2: Distribution of the precision achieved by different variants of the **DE** algorithm on $5D$ problem instances for different modular configurations, across different budgets. The precision values are inversely proportional to algorithm performance, with smaller values indicating better performance.

We have also conducted the same empirical analysis for the 30D problem instances, and the results are available in our Zenodo repository [227]. Similar observations can be made for the 30D problem instances as in the case of using the 5D problem instances.

6.5.1.2 The importance of the ELA features for different modular setups

Following our experimental design, to obtain the Shapley-based meta-representations (i.e., landscape feature importance scores), we first trained separate regression models for each algorithm variant. Table 6.3 presents the average R^2 scores of the RF and baseline regression models for the CMA-ES and DE algorithm variants, for both $5D$ and $30D$ problems and the six different function evaluation budgets. Additionally, Table 6.4 summarizes the MSE scores for these models. As a baseline, we employ a model that consistently predicts the overall mean algorithm performance. One interesting pattern that can be observed is

that the models perform better for the $30D$ problems. This improved performance may be due to variations in the distributions of the target variable across different combinations of budget and dimensionality. The different distribution characteristics, such as skewness, can impact the performance of the RF model. Additionally, in a higher-dimensional space, where the points are spread out, certain ELA features might converge to specific values which can make the data simpler and easier for models to learn from.

Subsequently, we have utilized the SHAP algorithm to determine the feature importance of each of the 46 ELA features at the problem instance level. In the outer loop of the nested cross-validation, we have employed a leave-one-group-out CV validation with five groups (four for training and one for testing). However, we specifically focused on the Shapley values of the training folds, as this data is used to learn the predictive models and provides insight into the algorithms' workings.

Table 6.3: The R^2 scores of the RF regression models / R^2 scores of the baseline regression models averaged over the CMA-ES and DE algorithm variants for the BBOB problem instances in 5 and 30 dimensions where the target precision is measured at $B \in \{50D, 100D, 300D, 500D, 1000D, 1500D\}$ function evaluations.

Budget	CMA-ES		DE	
	$5D$	$30D$	$5D$	$30D$
$50D$	0.7577/-0.0072	0.9400/-0.0005	0.8788/-0.0019	0.9403/-0.0009
$100D$	0.7689/-0.0069	0.9179/-0.0008	0.8783/-0.0017	0.9433/-0.0008
$300D$	0.6146/-0.0072	0.8457/-0.0031	0.8587/-0.0016	0.9362/-0.0013
$500D$	0.7045/-0.0055	0.8322/-0.003	0.8368/-0.0024	0.9361/-0.0015
$1000D$	0.7272/-0.0046	0.8072/-0.0029	0.7795/-0.0043	0.9242/-0.002
$1500D$	0.7288/-0.0048	0.8391/-0.0023	0.7508/-0.0051	0.9191/-0.0023

Table 6.4: The MSE scores of the RF regression models / MSE scores of the baseline regression models averaged over the CMA-ES and DE algorithm variants for the BBOB problem instances in 5 and 30 dimensions where the target precision is measured at $B \in \{50D, 100D, 300D, 500D, 1000D, 1500D\}$ function evaluations.

Budget	CMA-ES		DE	
	$5D$	$30D$	$5D$	$30D$
$50D$	0.7829/4.0248	0.1482/2.461	0.3716/3.383	0.2642/3.7964
$100D$	1.2195/5.1834	0.2692/3.2073	0.4326/3.9106	0.2458/4.225
$300D$	3.9828/8.9112	1.0303/5.8091	0.809/5.6781	0.3055/4.9577
$500D$	4.8498/13.9604	1.2684/6.6078	1.0403/6.1126	0.3803/6.1386
$1000D$	5.2191/15.2566	1.7192/7.9612	1.9462/6.8843	0.5378/7.0673
$1500D$	5.1945/15.198	1.8771/10.9973	2.3355/7.4111	0.634/7.6636

To generate a Shapley value for each ELA feature and problem instance, we calculated the value four times (due to each problem instance appearing four times in the training data and once in the testing data) and then took the mean of the four values. Lastly, we averaged the Shapley values for each ELA feature across all problem instances, which gave us a single vector for each algorithm instance. For this purpose, we have leveraged TreeSHAP. TreeSHAP is tailored for tree-based models such as decision trees, random forests, and boosting machines. It is designed to be computationally efficient by exploiting the tree structure for faster calculations, which enables it to manage more complex scenarios effectively. One of the key advantages of TreeSHAP is its consistency property: if a model relies more on a particular feature, the attributed importance of that feature will not decrease, ensuring reliable feature attribution. Alternatively, KernelSHAP can be used for interpreting the impact of features in any model, as it employs a model-agnostic approach. While KernelSHAP offers flexibility across various model types, it comes at the cost of computational efficiency. This makes KernelSHAP less suitable for complex, high-dimensional situations or applications requiring real-time explanations. Given that we are working with tree-based predictive models, TreeSHAP was the appropriate choice for our study. It provided the necessary computational efficiency. The calculation of TreeSHAP is detailed in [11], where it is demonstrated that Shapley values can be used to interpret model performance regardless of whether the model performs well or poorly.

To investigate the importance of the ELA features, we conduct exploratory analysis by selecting the top K most important features, where K is chosen from the set $\{10, 15, 20\}$. Next, we tally the frequency of appearance for each feature in the top K across all algorithm variants within the same group of modular algorithm variants. This is done by calculating the number of times a feature appears in the top K as indicated by their Shapley values. The resulting value ranges between 0 and the total number of algorithm variants in the group.

Analyzing the results in Figure 6.3 and through our analysis of feature importance in various other scenarios, we have observed that a similar set of ELA features are the most important predictors of performance, regardless of the algorithm, modular configuration, problem dimensionality, or function evaluation budget. This suggests that we can perform feature selection for all algorithms simultaneously, irrespective of their configurations. However, we recommend training separate regression models for each algorithm configuration to obtain the most accurate predictions.

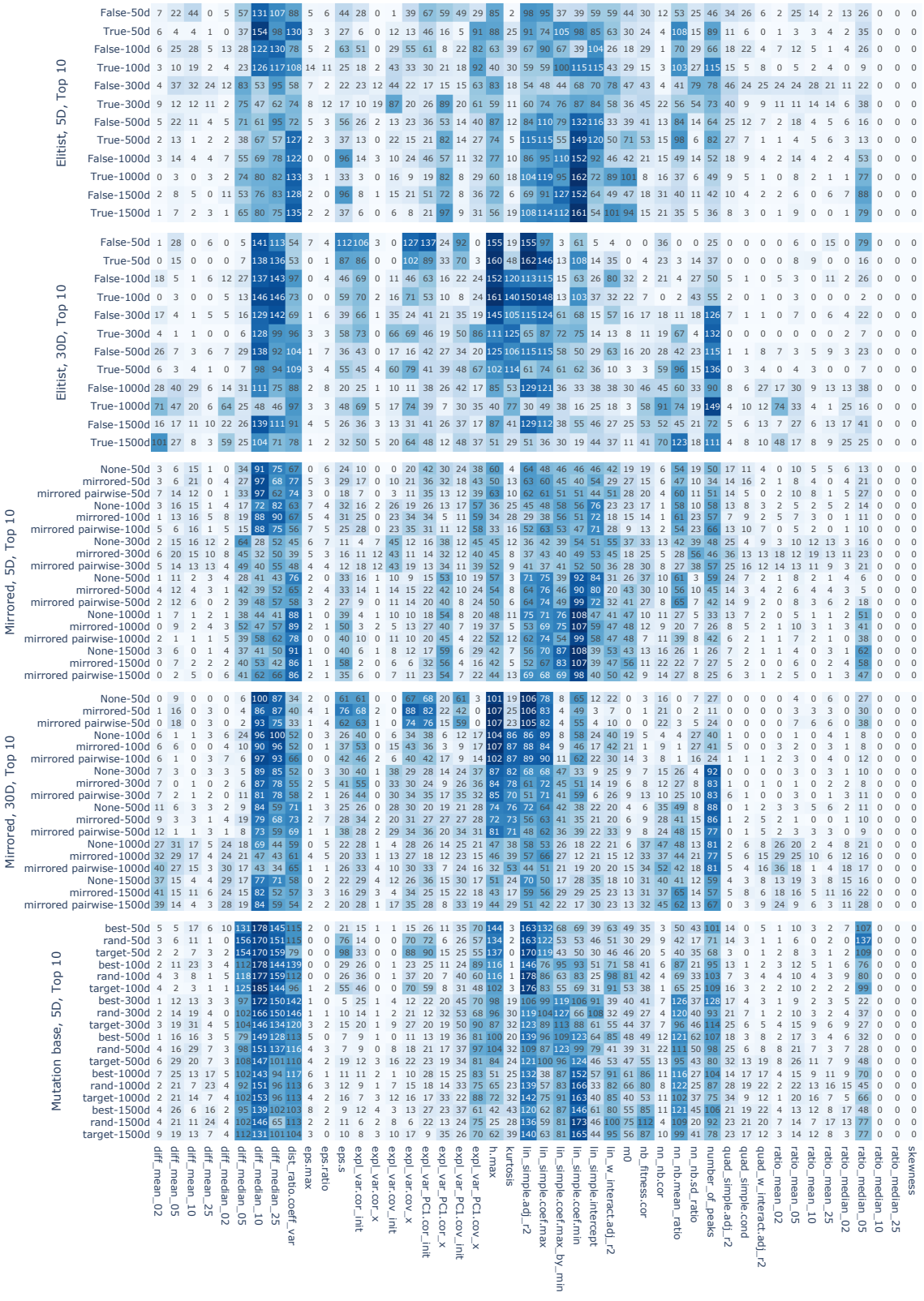


Figure 6.3: Frequency of appearance of the ELA features as top 10 most important features for performance prediction of two modCMA-ES modules (elitist and mirrored; first four groups) on the 24 BBOB functions in both 5 and 30 dimensions and for six different evaluation budgets $B \in \{50D, 100D, 300D, 500D, 1\,000D, 1\,500D\}$. The fifth group provides the same data for one DE module (mutation base).

6.5.2 Predicting the modular configuration of an algorithm using its behavior meta-representation

After exploring the performance and ELA data on which the meta-representations are built, we now analyze whether these meta-representations are powerful enough to predict/identify the corresponding algorithm variant.

First, we compare the single- and multi-output approaches for training classifiers using the RF method. The F1 scores for the classifiers obtained on the test data aggregated across the 2 problem dimensionalities, 5 budgets, and algorithm modules, are listed in Table 6.5. We have observed that comparable results can be obtained when employing single-output and multi-output RF techniques on both CMA-ES and DE algorithms, using both performance- and Shapley-based meta-representations. However, it is worth noting that the multi-output RF approach exhibits slightly inferior performance as compared to single-output RF.

Table 6.5: The F1 scores of the single-output RF, multi-output RF, and single-output TabPFN models, computed by averaging over the CMA-ES and DE algorithm variants. The F1 scores are further averaged for both 5 and 30 dimensions, and across the 5 budgets for function evaluation ($B \in \{50D, 100D, 300D, 500D, 1000D, 1500D\}$).

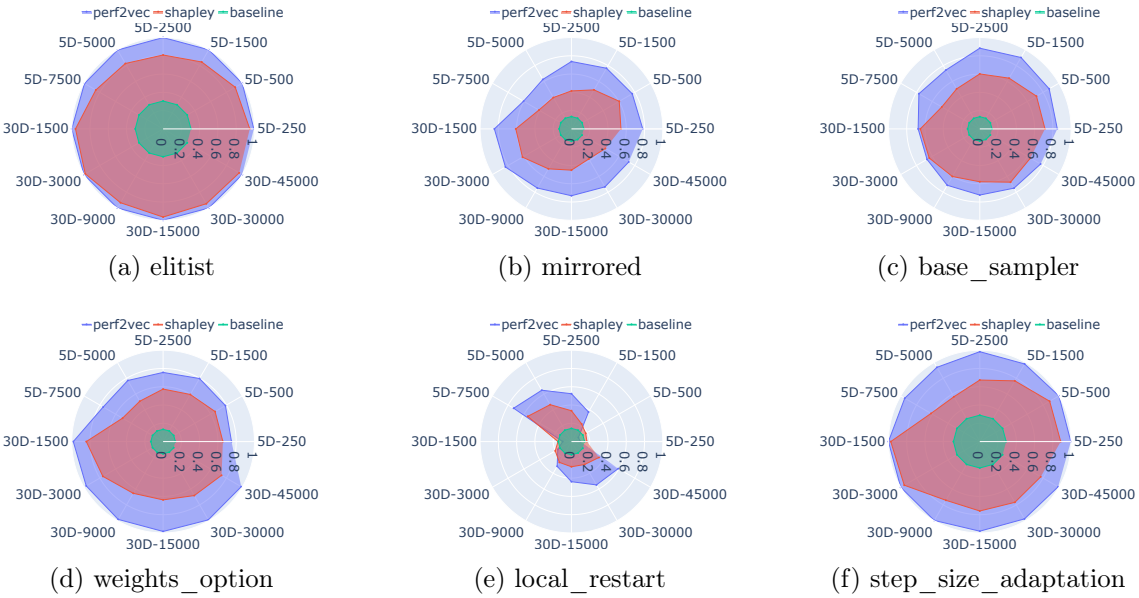
Algo	Performance			Shapley		
	single-output RF	multi-output RF	single-output TabPFN	single-output RF	multi-output RF	single-output TabPFN
CMA-ES	0.794	0.772	0.811	0.623	0.618	0.629
DE	0.758	0.744	0.790	0.603	0.601	0.589

Additionally, we have compared the performance of single-output RF classifiers with TabPFN classifiers. Both classifiers showed similar F1 scores, as indicated in Table 6.5. For instance, when using Shapley-based meta-representations, the RF classifier's average F1 score for predicting the modular configuration of CMA-ES variants was 0.623, while the TabPFN classifier's F1 score was 0.629, indicating slightly better performance for TabPFN. However, for the DE variants, we observed the opposite situation, with RF classifiers achieving an average F1 score of 0.603 and TabPFN of 0.589.

In Table 6.5, the F1 scores are averaged over all algorithm modules. To further analyze the classifiers trained using the single-output RF method, in Figure 6.4 we present the F1 scores of the classifiers for each CMA-ES module separately. As a baseline, we use the majority classifier. Figure 6.4 shows that the highest performance scores are achieved in predicting the setting of the elitist and step-size adaptation modules. The higher F1 scores for the elitist and step-size adaptation modules compared to the other four CMA-ES modules are expected because we only investigated two module options for these two modules, while the remaining four modules used three different module options. By having fewer classes to distinguish between, the classification problem is simplified, making it easier to solve.

The highest F1 scores among the remaining four modules have been observed for the weights option, followed by the base sampler and mirrored. The configuration of the local restart module is the most difficult to predict. In general, all classifiers for predicting the status of each module outperform the baseline across the different modules, problem dimensions, and budgets (see Figure 6.4).

Further, we have observed that the performance-based meta-representations have better predictive power than the Shapley-based meta-representations.



(a) elitist (b) mirrored (c) base_sampler

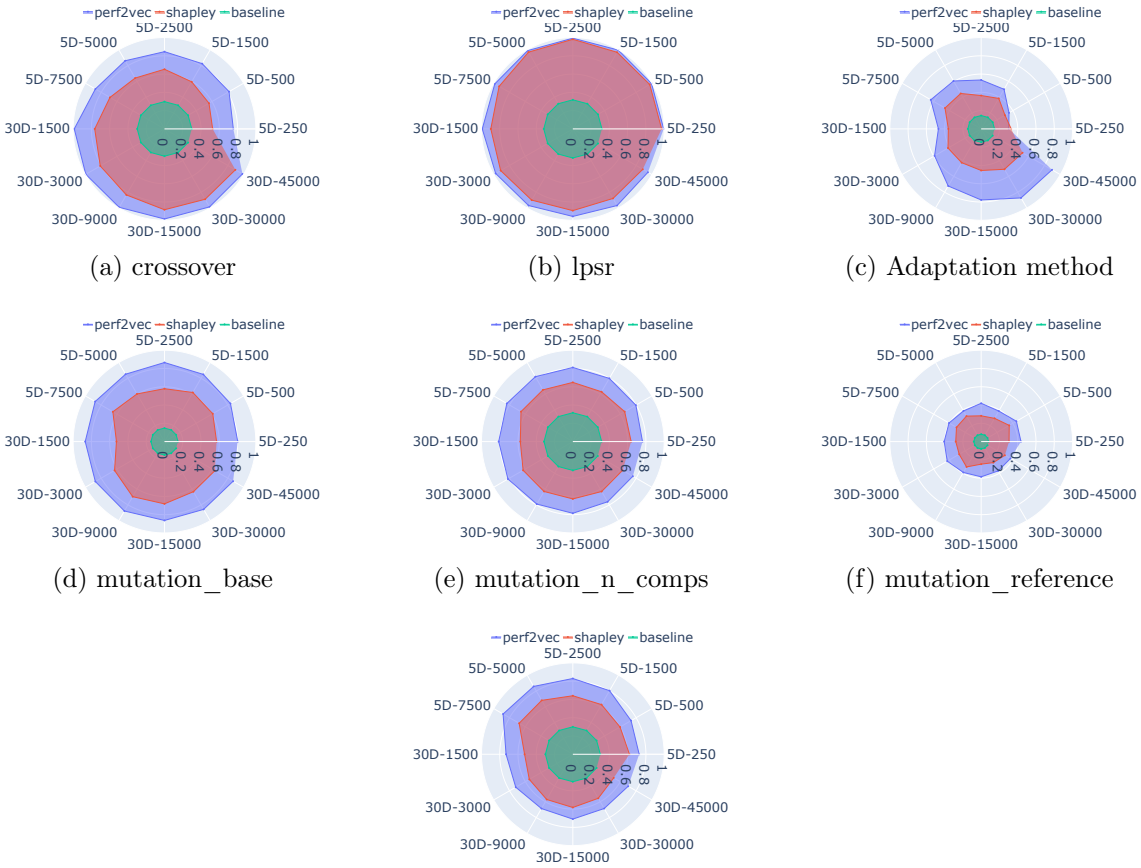
(d) weights_option (e) local_restart (f) step_size_adaptation

Figure 6.4: The F1 scores of the RF classifiers for predicting the modular configuration of the CMA-ES algorithm variants. Results are presented for each CMA-ES module separately, for $5D$ and $30D$ BBOB problem instances, and for 5 different function evaluation budgets. The baseline is the majority classifier.

In Figure 6.5, we show the F1 scores of the RF classifiers for each DE module. For the `mutation_n_comps`, `use_archive`, `crossover`, and `lpsr` modules we have considered two different module options. As can be seen in Figure 6.5, for these four modules the classifiers have the highest F1 scores, with `lpsr` classifiers performing the best, followed by `crossover`, `mutation_n_comps`, and `use_archive`. As the number of considered modular options increases (three different options for the mutation base and adaptation method modules and four different options for mutation reference), the F1 scores of the classifiers tend to decrease. For both CMA-ES and DE, it is worthwhile to note that the modules that have limited initial impact (local-restart and adaptation mechanism) are indeed more challenging to predict, especially for small budgets. Nevertheless, in all cases, the classifiers outperform the baseline classifier. Furthermore, the RF classifiers that used performance-based meta-representations consistently outperformed those that used Shapley-based meta-representations.

Performance difference between algorithm variants. By combining the predictions of the RF classifiers for all modules, we can predict the modular configuration of the algorithm instance. To judge the effectiveness of this prediction, we analyze the difference in performance between the true and predicted modular configurations. Even though there might be a difference in the configuration, the true and the predicted algorithms may have similar performance behavior. To evaluate this, we use the DSC approach to test for statistical significance in the performance of the true and the predicted configuration across all benchmark problem instances. First, we apply the DSC ranking scheme that ranks the true and the predicted configuration by comparing the distribution of their raw performance data for each problem instance separately.

For comparing the distributions, the two-sample Anderson-Darling test [228] is used by the DSC ranking scheme. Since most of the statistical tests require the independence



(a) crossover (b) lpsr (c) Adaptation method

(d) mutation_base (e) mutation_n_comps (f) mutation_reference

(g) use archive

Figure 6.5: The F1 scores of the RF classifiers for predicting the modular configuration of the DE algorithm variants. Results are presented for each DE module separately, for $5D$ and $30D$ BBOB problem instances, and for 5 different function evaluation budgets. The baseline is the majority classifier.

condition, we have aggregated the rankings per problem class by calculating the average of the DSC rankings obtained for the five problem instances that belong to that problem class. Next, the ranked data is analyzed with a statistical test. The rankings obtained for the 24 problem classes have been analyzed with the Wilcoxon signed-ranks test to find if there is a statistically significant difference (at a p-value of 0.05) in the performance of the true and predicted configuration on the selected benchmark suite. After determining the statistical significance of the difference between each true and predicted algorithm pair, we calculate the percentage of pairs with performance differences that are not statistically significant. Additionally, we generate five different random predictions for the modular configuration of each algorithm instance and perform the DSC analysis on them. The results for CMA-ES and DE across different problem dimensions, budgets, and meta-representations are shown in Table 6.6 and Table 6.7, respectively.

In both Table 6.6 and Table 6.7, we can observe that the percentage of algorithm pairs, consisting of true and predicted configurations, with performance differences that are not statistically significant (based on the predictions generated by our classifiers) is significantly higher as compared to the scenario where predictions are randomly generated for the modular configuration. This affirms the robust predictive capabilities exhibited by our classifiers.

Table 6.6: The DSC results on the statistical difference in the performance of **CMA-ES** algorithm pairs. Results are reported in the format: percentage of (true, predicted)-pairs with performance differences that are not statistically significant/percentage of (true, random)-pairs with performance differences that are not statistically significant. The numbers in brackets correspond to the standard deviation for the latter percentage since this was computed over 5 independent runs.

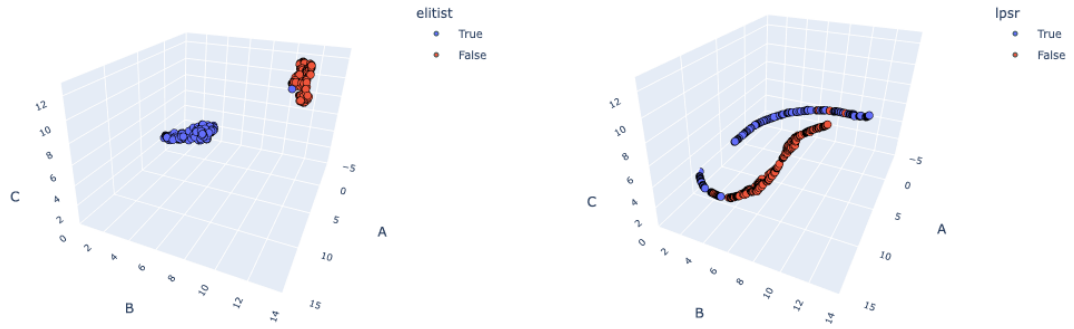
Budget	SHAP		Performance	
	$5D$	$30D$	$5D$	$30D$
$50D$	74.4 / 33.6 (2.0)	68.2 / 24.6 (2.7)	89.2 / 36.4 (2.0)	93.2 / 22.5 (1.2)
$100D$	80.9 / 62.2 (3.5)	68.2 / 26.9 (2.3)	91.0 / 64.0 (1.6)	93.5 / 27.0 (3.1)
$300D$	75.0 / 57.5 (1.6)	59.6 / 35.2 (3.0)	87.3 / 58.4 (3.1)	88.9 / 36.5 (1.7)
$500D$	67.6 / 57.2 (2.3)	55.2 / 33.5 (2.4)	89.5 / 55.8 (3.1)	83.6 / 34.8 (1.5)
$1000D$	74.1 / 48.7 (2.3)	54.3 / 36.8 (1.1)	84.9 / 48.5 (0.6)	82.7 / 38.9 (1.7)
$1500D$	72.2 / 43.6 (1.7)	49.1 / 34.0 (2.1)	88.0 / 42.7 (3.0)	81.2 / 34.1 (2.9)

Table 6.7: The DSC results on the statistical difference in the performance of **DE** algorithm pairs. Results are reported in the format: percentage of (true, predicted)-pairs with performance differences that are not statistically significant / percentage of (true, random)-pairs with performance differences that are not statistically significant. The numbers in brackets correspond to the standard deviation for the latter percentage since this was computed over 5 independent runs.

Budget	SHAP		Performance	
	$5D$	$30D$	$5D$	$30D$
$50D$	50.5 / 14.5 (1.2)	21.7 / 12.2 (1.5)	77.1 / 12.6 (1.7)	53.6 / 10.8 (0.7)
$100D$	45.0 / 11.2 (0.8)	23.6 / 10.9 (1.6)	70.5 / 12.2 (1.3)	54.3 / 11.7 (1.1)
$300D$	31.9 / 10.0 (1.4)	27.3 / 12.3 (0.7)	57.1 / 9.9 (1.1)	45.3 / 13.2 (0.9)
$500D$	31.1 / 11.5 (0.9)	28.6 / 16.6 (1.2)	56.9 / 11.3 (1.6)	48.1 / 15.7 (1.1)
$1000D$	31.2 / 13.0 (1.1)	31.8 / 19.5 (1.5)	49.5 / 12.6 (1.0)	45.5 / 19.0 (1.0)
$1500D$	33.0 / 16.0 (1.5)	30.4 / 23.2 (0.9)	54.5 / 15.0 (1.7)	46.5 / 22.5 (1.8)

An additional noteworthy observation is that the percentage of pairs with performance differences that are not statistically significant is higher for the CMA-ES algorithm variants as compared to DE. To further investigate this observation, we use UMAP [229] as a dimensionality reduction technique to depict the performance-based meta-representations of CMA-ES and DE algorithm variants. Specifically, we focus on the $5D$ problems and $300D$ budget cut-off.

Figure 6.6 showcases the UMAP plots, allowing for a visual examination of the performance space. Notably, the CMA-ES algorithm variants exhibit closer proximity to one another, forming two distinct clusters. This close grouping suggests similar performance characteristics among these variants. We have observed that the elitism module almost perfectly separates the algorithm variants into two clusters. In contrast, the DE algorithm variants display more pronounced differences in performance, leading to a lower percentage of pairs with performance differences that are not statistically significant. In this case, the purest clusters are formed by taking into consideration the configurations of the `lpsr` module, indicating that this module exerts the greatest influence on performance as compared to the other modules.



(a) CMA-ES (b) DE

Figure 6.6: UMAP embeddings of the performance-based meta-representations of the 324 CMA-ES and 576 DE algorithm variants.

6.6 Summary

In this study, we have proposed a methodology for examining the impact of different modules of optimization algorithms on the overall algorithm performance. We have demonstrated its relevance within two pre-existing modular optimization frameworks, namely modCMA-ES and modDE. To this end, we have analyzed performance data from 324 modCMA-ES and 576 modDE algorithm variants across 24 noiseless BBOB problems. Among the investigated CMA-ES modules, we have found that the elitism module has the most pronounced influence on performance, while the local restart module has the smallest influence, particularly for smaller runtime budgets. These findings are aligned with existing work analyzing these algorithms. Regarding DE, out of the seven modules examined, we have observed that the linear population size reduction module exerts the greatest influence on performance. The mutation reference and adaptation method modules have considerably smaller effects as compared to the other modules.

Although our findings on some modules are not conclusive, our methodology is adaptable and can be applied to other modular optimization frameworks, where it may yield different insights. The methodology to other modular optimization frameworks, such as ParadisEO [80], [81], PSO-X [82] and the modular hybridization framework of particle swarm optimization and differential evolution [83].

Observing variations in the impact of different modules on performance lead us to conclude that to accurately assess the contribution of a new idea or algorithm design, it is crucial to compare algorithm modules rather than algorithms themselves.

Furthermore, we have trained classifiers to predict the modular configuration of algorithm variants. We have found that the classifiers achieve higher F1 scores, in both cases of using performance-based and Shapley-based algorithm meta-representations when the impact of the module on performance is more substantial. This observation was expected because, in cases where the impact is less significant, the algorithm variants tend to be closer in the meta-representation space, making it challenging for the ML model to differentiate effectively.

Classifiers using performance-based meta-representations show superior predictive performance compared to those built from Shapley-based meta-representations. However, it is worth noting that performance-based meta-representations are less flexible when it comes to accommodating new problem classes, as the vector size is predetermined by the number of classes. On the other hand, Shapley-based meta-representations, which rely on problem

landscape features, maintain a consistent vector size when introducing new problem classes. Nonetheless, a limitation arises when new classes are introduced, requiring the retraining of regression models used to determine the Shapley-based landscape feature importance.

With respect to the importance of the landscape features, it seems that the same ELA features appear to be the most important features that contribute to the performance of the algorithm performance prediction regression models, regardless of the possible values of each module.

Chapter 7

Predicting Algorithm Performance in Numerical Black Box Optimization with Knowledge Graph Reasoning

In this chapter, we exploit benchmarking data about modular optimization algorithms, and investigate the use of formal semantic representations to predict the performance of black-box optimization algorithms. Specifically, we focus on the feasibility of using classical knowledge graphs, which do not rely on node features or additional data, to predict the performance of algorithms from two modular frameworks, modCMA-ES and modDE. The task is framed as a knowledge graph completion problem, where the goal is to infer missing relationships within the graph.

The data is represented in a factual, discrete form, capturing semantic relationships between entities such as algorithms, problem instances, and their performance. To explore this type of representation, we employ a scoring-based knowledge graph embedding (KGE) model. By training KGEs, we aim to predict performance links (*solved* or *not-solved*) between algorithm configurations and problem instances under specific precision thresholds in a fixed-budget scenario.

Section 7.1 provides a detailed problem definition and discusses the potential of KGs for predictive modeling in black-box optimization. Section 7.2 outlines the methodology and experimental setup, describing the construction of the KG and the KG embedding-based pipeline for algorithm performance prediction. In Section 7.3, we present the experimental results from different evaluation scenarios, including both balanced and imbalanced classification settings. Finally, Section 7.4 summarizes our findings, discusses the limitations of the transductive learning setting, and outlines directions for future research.

This chapter is based on the paper "Using Knowledge Graphs for Performance Prediction of Modular Optimization Algorithms" [230], which appeared in the International Conference on the Applications of Evolutionary Computation (Part of EvoStar), 2023.

All data and code related to this chapter are publicly available on GitHub at: `https://github.com/KostovskaAna/KG4AlgorithmPerformancePrediction`.

7.1 Problem Definition

A KG is a set of relational facts represented by entities and the relationships between them [231]. KGs have become invaluable tools for organizing and leveraging complex relationships within data across various domains. Their ability to integrate heterogeneous information and infer new insights has led to successful applications in numerous fields.

For example, in natural language processing, KGs enhance search engines and recommendation systems by providing context and relational data [232], [233]. In the biomedical field, they assist in drug discovery and disease research by integrating diverse biological data sources [234], [235]. KGs are also used in social network analysis to understand and predict user behavior [236], [237].

In machine learning predictive modeling, KGs have demonstrated significant potential. One study used a KG with nodes for drugs, protein targets, indications, and adverse reactions to predict unknown adverse drug reactions, outperforming standard ML techniques [238]. Another study utilized a KG built from National Health and Nutrition Examination Survey data for health risk prediction, achieving superior performance compared to baseline classifiers [239].

The success of KGs in these diverse domains highlights their potential for addressing complex challenges in other fields as well. In evolutionary computation research, empirical analysis and benchmark studies on iterative black-box optimization algorithms play a crucial role. The data involved in these studies is often complex and heterogeneous. Optimization problems, for instance, can be represented with numerical vectors that capture landscape characteristics, while high-level problem features can be represented with categorical values (e.g., separability, multi-modality) [54]. There is also considerable diversity in optimization heuristics, which stems from variations in operator choices, parameter adjustment strategies, and hyperparameter settings. The different parameters and hyperparameters of the optimization algorithms influence various aspects of the optimization process, such as initialization, mutation, and recombination. These variations can result in significantly different algorithm behaviors [240]. Additionally, performance data can be represented using various metrics, further adding to the complexity of analyzing and comparing algorithm performance [241].

Given the demonstrated effectiveness of KGs in managing and leveraging complex, heterogeneous data across various domains, applying KGs to evolutionary optimization algorithms presents a promising research opportunity. KGs can capture the intricate relationships between diverse data elements, providing a unified framework for understanding and analyzing the optimization process.

By explicitly representing entities such as problems, algorithms, and parameters as nodes, and their interactions and dependencies as edges, KGs facilitate the integration, visualization, and exploration of complex data. This structured representation enables researchers to examine the relationships and interactions within the optimization process.

Additionally, KGs can be leveraged for predictive modeling and reasoning tasks. In black-box optimization, predictive models built using KG data can forecast algorithm performance and identify optimal configurations by learning from the rich, interconnected data. Furthermore, reasoning tasks such as Knowledge Graph Completion (KGC) can infer missing relationships or entities, enhancing the utility of KGs by uncovering hidden patterns and insights. This capability can lead to more informed decision-making and improved optimization outcomes.

In Chapter 3, we introduced the OPTION ontology [130], designed to standardize data representation in the domain of single-objective numerical optimization. The corresponding OPTION KB developed is an RDF-based repository that provides structured, semantically enriched storage, making complex data more accessible for researchers. This semantically annotated data forms an RDF graph, which can also be viewed as a KG, enabling efficient querying and retrieval. Additionally, by leveraging the OPTION ontology, it enhances interoperability across different systems and platforms, ensuring seamless data sharing and understanding.

Although there have been various efforts to organize knowledge in black-box optimiza-

tion, these studies have mainly focused on representing and structuring domain knowledge [139]–[141], [242], [243]. This is also true for OPTION, where we have concentrated on organizing and representing knowledge and data. As far as we know, these semantic data representations have not yet been tested for their effectiveness in analytical and predictive studies.

While evolutionary optimization methods have been used to improve knowledge graphs [244]–[246], knowledge graphs have not yet been applied directly to evolutionary optimization algorithms. This gap presents a promising opportunity for future research to explore how KGs can be integrated with evolutionary optimization algorithms, potentially unlocking new ways to solve complex optimization problems.

While the RDF data model (used for the development of the OPTION KB) helps create graph structures and is the most popular for semantic data management in academic domains, it is complex and verbose, limiting the effectiveness of the use of graph-based analysis techniques. RDF often includes abstract entities for OWL axioms, implicit statements, and complex N-ary relationships with provenance information, adding to its complexity.

Recognizing this, in this chapter, we propose a transition from an RDF-based OPTION KB to a OPTION KG that is more accessible for applying graph-based algorithms. The KG will provide a simpler and clearer view by focusing on entities and their connections, making it easier to interact with the data and apply various graph algorithms. Each of the entities in the KG will have a one-to-one mapping to OPTION ontology classes and instances, ensuring that the semantic structure is preserved.

In this chapter, we propose a novel approach that leverages the OPTION ontology and KGs for predicting algorithm performance. We will employ a custom KG view over our OPTION KB to perform knowledge graph reasoning, with a focus on KG completion, one of the most common reasoning tasks. Specifically, we will represent optimization problems, their descriptors, algorithms, and their configurations as nodes. By performing a link prediction task using Knowledge Graph Embeddings (KGEs) [247], we aim to predict the missing links between problem and algorithm nodes, which represent the performance the algorithm achieves on a given problem.

KGEs are low-dimensional, feature-based representations of the entities and relationships in a knowledge graph. They provide a generalizable context across the entire KG, enabling tasks such as KG completion, triple classification, link prediction, and node classification [231]. This approach would enhance our ability to understand and utilize the interconnected data within the KG.

7.2 Methodology and Experimental Setup

In this chapter, we investigate the use of KGEs to predict algorithm performance through KGC. Our objective is to predict unseen performance relationships between problem instances and algorithm configurations, determining whether an algorithm can achieve a specified target precision for a given problem. This task can be formulated as a binary classification problem.

7.2.1 Knowledge graph completion for automated algorithm performance prediction

We perform the KGC task by predicting missing links, referred to as performance links, between problem instances and algorithm configurations. Specifically, our task is to classify whether an algorithm configuration will achieve a specified target precision for a given problem instance. Although algorithm performance prediction is inherently a regression

task, the symbolic nature of data in KGs necessitates converting this regression task into a classification problem.

To achieve this, we set different target precision thresholds and predict whether the algorithm has reached a solution quality that meets each threshold. The solution quality is measured by the precision achieved by the algorithm, defined as the distance from the optimum. This transformation allows us to work effectively within the symbolic data framework of KGs.

We focus on modular frameworks as these frameworks provide a structured representation of algorithms suitable for KG modeling.

7.2.2 Construction of the knowledge graph

The primary node types in the KGs are problem instances and algorithm instances (see Figure 7.1). We use the same benchmarking data as described in Chapter 6. This includes the first five instances of each of the 24 noiseless BBOB problems [54] in dimensions $D = 5$ and $D = 30$, resulting in two problem sets (one for each dimension) with 120 problem instances each. Each problem instance is characterized by high-level and low-level landscape features. The high-level features include five problem classes (i.e., separable, low or moderate conditioning, high conditioning and uni-modal, multi-modal with adequate global structure and multi-modal with weak global structure) introduced in the BBOB test suite [54], which group benchmark problems with similar properties. The low-level landscape features consist of 46 ELA metrics. For details on their calculation, refer to Section 6.4.1. Each ELA feature is discretized into 10 bins to follow the symbolic representation framework of KGs.

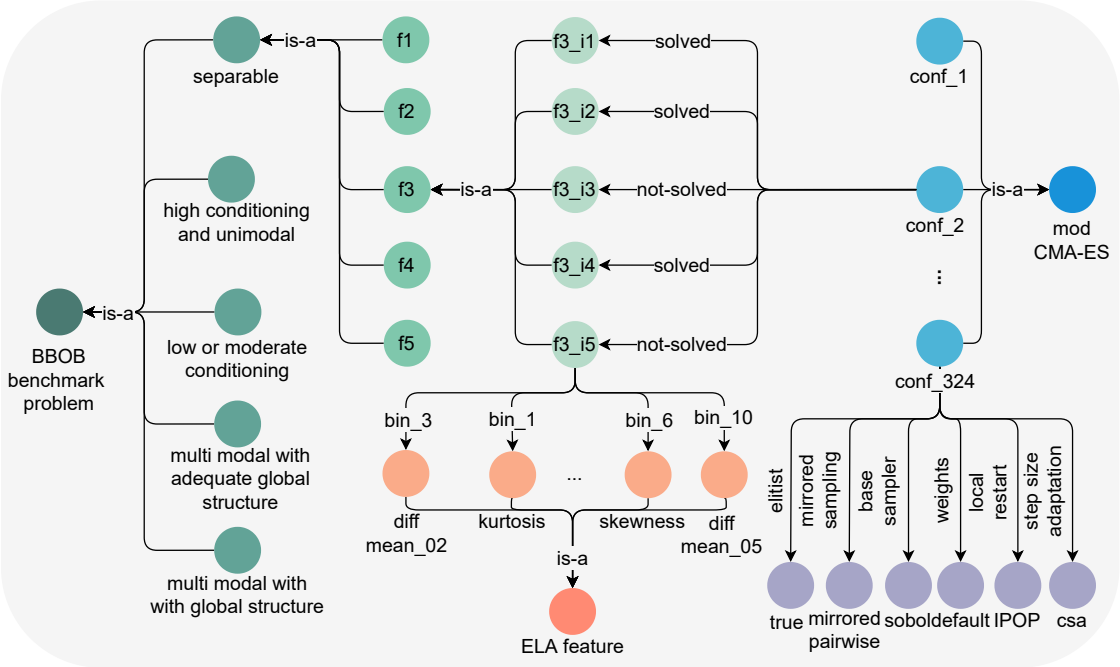


Figure 7.1: A snippet of a knowledge graph visualizing the representation of problem instances, including their high-level and low-level feature representations, as well as the algorithm instances linked to their respective configuration setups.

The algorithm instances are derived from two distinct modular algorithms, modCMA-ES and modDE, which were investigated in the preceding chapters. Specifically, we utilize

324 algorithm instances for modCMA-ES and 576 instances for modDE, representing diverse configurations and variants for analysis. Section 4.3.5.5 provides detailed information on these configurations, which are available via the OPTION KB. In our KGs, each algorithm instance is represented as a node and is connected to the different modules via labeled links/edges. Performance data is also detailed in Section 4.3.5.5. As a performance measure, we use the target precision achieved by the algorithm within a fixed budget (i.e., after a specified number of function evaluations), using the best precision achieved after $B = \{2\,000, 5\,000, 10\,000, 50\,000\}$ function evaluations.

Finally, problem instances and algorithm configurations are linked with either a *solved* or *not-solved* edge, depending on the algorithm's performance against three different target precision thresholds: $T = \{1, 0.1, 0.001\}$ for the $5D$ benchmark problems and $T = \{10, 1, 0.1\}$ for the $30D$ benchmark problems. Specifically, if an algorithm instance achieves a target precision equal to or lower than the specified threshold for a given problem instance, we link the algorithm instance and the problem instance with a *solved* edge; otherwise, we link them with a *not-solved* edge.

We create a separate KG for each combination of budget, target precision, problem dimension, and algorithm family. With four budgets for function evaluations, three target precisions, two problem dimensions, and two algorithm families, this results in a total of 48 KGs.

7.2.3 KG embedding-based pipeline for automated algorithm performance prediction

Our knowledge graph G is represented as a collection of triples $\{(h, r, t)\} \subseteq E \times R \times E$, where $h, t \in E$ and $r \in R$ are the entity and relation set. One of the tasks in KG completion is to predict unseen relations r between two existing entities $(h, ?, t)$. In this context, we focus on the $\{(a, s, p)\} \subseteq A \times S \times P$ triples, where $a \in A \subset E$ and $p \in P \subset E$ are the algorithm and the problem instance sets, respectively, and $S = \{solved, not\text{-}solved\} \subset R$ represents the performance relation. The goal is to predict the unseen performance relations between algorithm configurations and problem instances $(a, ?, p)$.

Our proposed pipeline for predicting algorithm performance is illustrated in Figure 7.2. It consists of two main parts: training the KG embeddings and using them in the inference phase to predict the type of a missing performance links.

7.2.3.1 Training phase

For training the KG embeddings, we utilize the Ampligraph library [248]. During the training phase, we initialize the KG embeddings with the Xavier initializer [249] and update them over several epochs.

This process incorporates a ComplEx scoring function [12], a model-specific function that assigns a score to each triple. ComplEx extends the DistMult [100] scoring function into the complex space, allowing it to model asymmetric relations effectively. Since ComplEx embeddings belong to $\mathbb{C}$, this model uses twice as many parameters as DistMult, providing greater representational capacity. Scoring functions for knowledge graph embeddings measure how far away two entities are relative to the relation in the embedding space. The primary goal is to maximize the scoring function for positive triples and minimize it for negative ones.

ComplEx scoring function is based on the trilinear Hermitian (or sesquilinear) dot product in $\mathbb{C}$:

$$f_{ComplEx} = Re(\langle \mathbf{w}_r, \mathbf{e}_h, \overline{\mathbf{e}_t} \rangle)$$

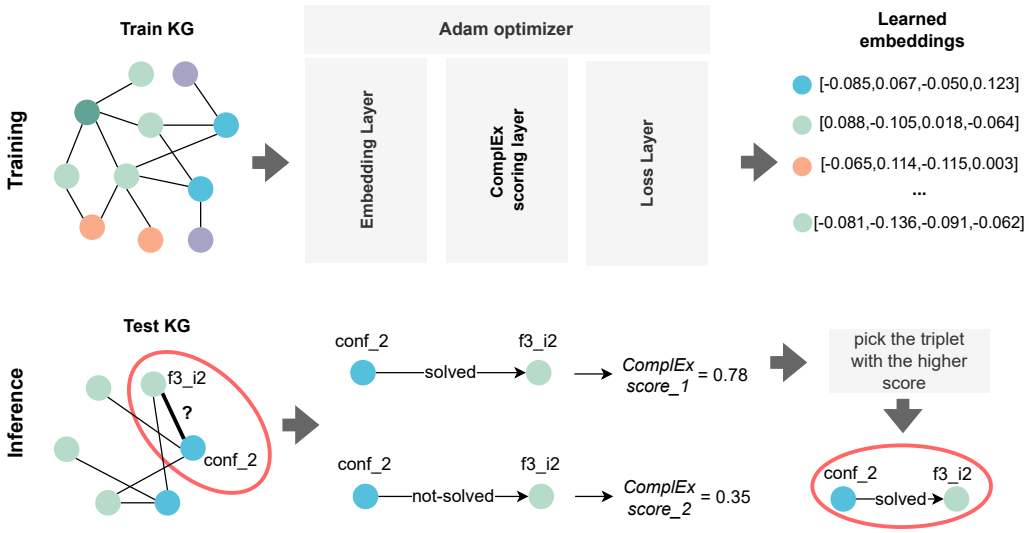


Figure 7.2: An illustration of the methodology for training the KG embeddings and the inference pipeline for automated algorithm performance prediction.

where $\mathbf{w}_r$, $\mathbf{e}_h$, and $\overline{\mathbf{e}_t}$ are the complex embeddings of the relation, head, and the complex conjugate of the tail, respectively. Re denotes the real part of the complex value, and $\langle \cdot \rangle$ represents the Hermitian dot product, which allows the model to capture asymmetric relationships.

This scoring function is then used on both positive and negative triples during the training process. To generate negative triples, we perform negative sampling on the knowledge graph by corrupting either the head (h) or the tail (t) part of the triples. Specifically, for a given positive triple (h, r, t) that is part of the KG, we create negative triples by replacing the head h with a randomly selected entity h' or the tail t with a randomly selected entity t', forming (h', r, t) and (h, r, t').

The training process aims to maximize the scores of positive triples and minimize the scores of negative triples. This is achieved by optimizing a loss function that incorporates contributions from both positive and negative triples. To perform the optimization, we utilize the Adam Optimizer [250].

7.2.3.2 Inference phase

During the inference phase, we predict missing performance relations for triples of the form $(a, ?, p)$. For each of these triples, we calculate the ComplEx model scores for both (a, solved, p) and $(a, \text{not-solved}, p)$ using the learned embeddings. The performance relation with the higher ComplEx score is selected as the inferred relation. This approach allows us to determine whether an algorithm configuration a solves a problem instance p based on the highest scoring triple.

7.2.3.3 Evaluation

The evaluation of the embeddings on a validation and test set employs the same inference methodology. For each (a, s, p) triple in the data set, where s is the true label (*solved* or *not solved*), we calculate the ComplEx model score for both (a, solved, p) and $(a, \text{not-solved}, p)$ triples using the learned embeddings. By comparing these scores, we predict the performance relation as the one with the highest score. This process is repeated for all triples in

the dataset. To assess the performance of our model in predicting the correct performance relations, we compute standard classification performance metrics, such as the F1 score.

7.2.3.4 Hyper-parameter tuning

To find the best hyperparameters for training the KGEs, we used the grid search methodology, which performs an exhaustive search over the selected hyperparameters and their corresponding search spaces. Three different hyperparameters were selected for tuning.

The first hyperparameter is k, the dimensionality of the embedding space, with values considered being 50, 100, 150, and 200. This parameter determines the size of the embedding vectors for the entities and relations in the knowledge graph.

The second hyperparameter is the optimizer's $learningrate$, with values considered being 1×10^{-3} and 1×10^{-4}. The learning rate controls the step size at each iteration while moving toward a minimum of the loss function.

The third hyperparameter is the type of $lossfunction$ used during training. We evaluated three different loss functions: the pairwise margin-based loss, which aims to maximize the margin between positive and negative triples; the negative log-likelihood (NLL) loss, which minimizes the log-probability of incorrect triples, thus favoring correct triples; and the self-adversarial sampling loss, improving robustness by focusing on hard negatives.

The optimal set of hyperparameters is estimated using a separate validation set. Initially, the number of training epochs is set to 500, but an early stopping mechanism can be activated to terminate training if 10 consecutive validation checks/epochs do not improve performance. The F1 metric is used as a heuristic in the grid search step.

7.3 Results and Discussion

In this section, we present the experimental results from two different evaluation scenarios based on the assignment of performance triplets to the training, validation, and testing sets. The first scenario employs what we term leave-random-performance-triplets-out validation 7.3.1, while the second scenario utilizes leave-problem/algorithm-instances-out validation 7.3.2.

7.3.1 Leave-random-performance-triplets-out validation

For our first set of experiments, we perform automated algorithm performance prediction using the method described in Section 6.3. Since we consider two problem dimensionalities, four function evaluation budgets, and three target precision thresholds, we have a total of 24 different KGs for each of the two algorithms (modCMA-ES and modDE).

For each of the KGs, we split the performance triples in the ratio 60:20:20. That is, 60% of the triples are assigned to the training set, 20% to the validation set, and the remaining 20% to the test set. We do this in a stratified fashion, keeping the distribution of performance links as in the original KG. Since the split is based on a stratified sample of the performance links, performance links related to a particular problem instance or algorithm configuration can be split between the training/validation set and the test set.

This approach can be applied when the performance of algorithm instances is known for the majority of problem instances in the selected problem portfolio but is unknown for some. It is crucial that the problem and algorithm portfolios remain fixed, with all problem and algorithm instances appearing in the training set due to the transductive nature of the KGE methods we consider. Note that the training KG contains not only the performance triples but also other types of entities and relations, such as high-level and low-level landscape features and descriptions of algorithm instances in terms of modules

Table 7.1: The percentage of *solved* links for the modCMA-ES and modDE algorithms in the KGs composed of a) $5D$ and b) $30D$ problems across the different budget and target precision thresholds.

	modCMA-ES				modDE			
	2000	5000	10000	50000	2000	5000	10000	50000
1	62.9	68.2	71.3	78.9	27.7	42.2	58.1	81.4
0.1	46.8	54.1	57.1	63.7	13.2	23.3	33.1	62.8
0.001	36.9	47.8	50.7	55.9	9.4	14.4	21.7	56.2

(a) $5D$ problems

	modCMA-ES				modDE			
	2000	5000	10000	50000	2000	5000	10000	50000
10	35.1	46.2	49.9	68.1	13.0	21.6	29.1	46.1
1	10.7	16.0	21.0	40.9	1.6	4.4	8.0	17.5
0.1	6.1	08.8	12.5	31.5	1.2	3.2	6.2	12.7

(b) $30D$ problems

and their parameters. The validation and test sets, however, contain only the links/triples of interest, specifically the 'solved' and 'not-solved' performance links.

The percentage of the 'solved' performance relations with respect to 'not-solved' ones for the modCMA-ES and modDE KGs for the KG composite problem in 5 and 30 dimensions are shown in Table 7.1. We can notice that in some of the scenarios, we are dealing with imbalanced classification, especially in the case of $30D$ problems.

Table 7.2 presents the F1 scores of the performance classifier and the percentage improvement compared to the baseline across different budget and target precision thresholds for both $5D$ and $30D$ problems. As a baseline, we used the classifier that predicts the majority class (solved/not-solved class). While the our classifier shows improved performance in balanced classification scenarios, it suffers in cases of imbalanced classification.

The ComplEx link prediction model suffers from imbalanced classification as it does not have enough examples of the minority class to learn effective representations. This lack of sufficient minority class examples results in a performance drop, indicating that our proposed pipeline requires adjustments to handle imbalanced data effectively.

7.3.2 Leave-problem/algorithm-instances-out validation

Our second set of experiments evaluates a scenario where there is no performance data for a given problem or algorithm instance. To assess the performance of our classifier in this setup, we investigate two evaluation scenarios:

- **Leave-problem-instances-out validation:** In this scenario, we use all performance triples of one problem instance from each of the 24 BBOB problems for testing, select the performance triples from another problem instance for validation, and use the remaining three for training. For example, we use the first three instances of each of the 24 BBOB problems for training, the fourth instance for validation, and the fifth instance for testing. We repeat this five times so that each of the five instances appears once in the test set.

- **Leave-algorithm-instances out validation**: In this scenario, the algorithm in-

Table 7.2: The F1 score and the percentage of improvement compared to the baseline of the modCMA-ES and modDE algorithm performance classifier obtained using the ComplEx scoring model for the KGs composed of $5D$ and $30D$ problems across the different budget and target precision thresholds.

	2000	5000	10000	50000
1	0.922/19.43%	0.942/16.15%	0.944/13.33%	0.953/8.05%
0.1	0.905/30.22%	0.933/32.91%	0.937/29.06%	0.942/21.08%
0.001	0.893/15.37%	0.944/37.61%	0.944/40.48%	0.946/31.75%

(a) $5D$ problems - modCMA-ES

	2000	5000	10000	50000
1	0.848/1.07%	0.876/19.67%	0.901/22.59%	0.946/5.46%
0.1	0.788/-15.18%	0.82/-5.53%	0.858/6.98%	0.922/19.43%
0.001	0.831/-12.62%	0.745/-19.20%	0.803/-8.65%	0.919/27.64%

(b) $5D$ problems - modDE

	2000	5000	10000	50000
10	0.937/19.06%	0.927/32.62%	0.939/40.78%	0.953/17.51%
1	0.902/-4.45%	0.808/-11.50%	0.855/-3.17%	0.929/25.03%
0.1	0.935/-3.41%	0.89/-6.71%	0.852/-8.68%	0.921/13.28%

(c) $30D$ problems - modCMA-ES

	2000	5000	10000	50000
10	0.9/-3.23%	0.931/5.92%	0.947/14.10%	0.948/35.24%
1	0.504/-49.19%	0.792/-19.02%	0.846/-11.69%	0.87/-3.76%
0.1	0.695/-30.08%	0.735/-25.30%	0.835/-13.74%%	0.885/-5.04%

(d) $30D$ problems - modDE

stances are split with a 60:20:20 ratio and their performance triples are selected for training, validation, and testing, respectively. In order to assess the robustness of the results, we repeat this procedure five times independently.

We have applied these evaluation scenarios to the KGs comprised of $5D$ benchmark problems and modCMA-ES algorithm instances across four different budgets with a target precision threshold of 0.1. The F1 scores of the classifier (averaged over five runs), their standard deviations, and the percentage of improvement are displayed in Table 7.3. Similar to Section 7.3.1, our approach shows improvement compared to the baseline in cases of balanced classification.

Table 7.4 presents the evaluation results for the modDE performance classifier, where similar patterns can be observed. We observe that the performance of the KG triple classifier drops compared to the predictive performance achieved in the leave-random-performance-triplets-out validation experiments. This drop in performance is expected, as no performance links of the problem/algorithm instances that are in the test set appear in the training and validation sets. In the first set of experiments (a classical scenario for KG completion), we randomly removed performance triples without considering which problem instances or algorithm instances they are associated with. Thus, performance triples for a given problem instance or algorithm instance can appear in both the training and test

Table 7.3: The F1 score and the percentage of improvement compared to the baseline of the **modCMA-ES** algorithm performance triple classifier for the KGs where all performance links are removed for a subset of problems and algorithm configurations composed of **5D** problems across the different budgets and a target precision threshold of **0.1**.

	Leave-problems-out	Leave-algorithms-out
2000	0.728 (0.006)/4.90	0.893 (0.009)/28.67
5000	0.761 (0.018)/8.40	0.915 (0.011)/30.34
10000	0.766(0.008)/5.36	0.91 (0.011)/25.17
50000	0.797(0.014)/2.44	0.913 (0.002)/17.35

Table 7.4: The F1 score and the percentage of improvement compared to the baseline of the **modDE** algorithm performance triple classifier for the KGs where all performance links are removed for a subset of problems and algorithm configurations composed of **5D** problems across the different budgets and a target precision threshold of **0.1**.

	Leave-problems-out	Leave-algorithms-out
2000	0.854(0.061)/-8.07%	0.79(0.035)/-14.96%
5000	0.837(0.022)/-3.57%	0.85(0.021)/-2.07%
10000	0.796(0.024)/-0.75%	0.825(0.010)/2.87%
50000	0.83(0.010)/7.51%	0.822(0.013)/6.48%

sets, which is an easier problem to learn.

7.3.3 Addressing the problem of imbalanced classification

To address the issue of imbalanced classification, we modify the pipeline described in Section 7.2.3. Specifically, after the KG training phase, we add an additional step where we train a Random Forest (RF) classifier based on the learned embeddings. Our data instances are the performance triples. To generate the data for the RF classifier, we represent each (a, s, p) triple as a concatenation of the embedding vectors of the a and p entities. During inference, we use the RF classifier instead of directly using ComplEx scores. A similar approach was used by [251], which used KGEs for drug-drug interaction prediction and represents. To represent feature vector of a drug pair, they concatenated embedding vectors of each drug in the pair and used classifiers such as Logistic Regression, Naive Bayes and Random Forest.

We evaluate this approach using the most imbalanced scenario from the experiments in Section 7.3.1, specifically the setup where we predict modDE performance on $30D$ problem instances with a target precision threshold of 0.1. We train the RF classifier with default hyperparameters, implemented in the scikit-learn library [185]. In Table 7.5, we compare the F1 scores of the classifiers trained using the pipeline presented in Section 7.2.3 with the scores of the RF classifiers described in this section. The results show that training an RF classifier on the learned embeddings improves performance in terms of the F1 score.

Since we are dealing with imbalanced classification, the choice of the evaluation measure is essential. In Table 7.6, we additionally report the AUC-ROC, average precision, and geometric mean scores. These metrics confirm that the embedding-based RF classifier improves the performance prediction method.

We believe that the results improve because there might be separability in the embeddings space that the RF models manage to capture when predicting the algorithm's

Table 7.5: Comparison of the two proposed pipelines for **modDE** performance prediction on the **30D** problem instances with a target precision of **0.1**. Results are reported in the format: F1-score of the classifier/F1-score of the baseline/Percentage of improvement compared to the baseline.

	KG - ComplEx scoring	RF classifier
2000	0.695/0.994/-30.08%	0.999/0.994/0.52%
5000	0.735/0.984/-25.30%	0.998/0.984/1.43%
10000	0.835/0.968/-13.74%	0.996/0.968/2.91%
50000	0.885/0.932/-5.04%	0.991/0.932/6.27%

Table 7.6: Performance of the **RF classifier** for **modDE** performance prediction on the **30D** problem instances with a target precision of **0.1**. Results are reported in the format: Performance of the classifier/Performance of the baseline.

	AUC ROC	Average precision	G-mean
2000	0.994/0.5	0.962/0.012	0.963/0.0
5000	0.998/0.5	0.975/0.032	0.962/0.0
10000	0.998/0.5	0.977/0.062	0.954/0.0
50000	0.987/0.5	0.964/0.127	0.947/0.0

performance. When you concatenate the embeddings of the problem and algorithm instances and train a Random Forest (RF) classifier, you leverage the strengths of both the embeddings and the RF algorithm. The RF classifier might be using the embeddings more effectively by focusing on the discriminative features that separate the "solved" and "not solved" classes, thus achieving better performance even in an imbalanced setting. However, this assumption requires further investigation.

7.4 Summary

In this chapter, we have investigated the predictive power of formal semantic representations for automated prediction of algorithm performance in black-box optimization. Specifically, we evaluated the feasibility of using KGs to predict the performance of the modCMA-ES and modDE optimization algorithms on the noiseless BBOB functions. Our goal was to determine whether we could train KG embeddings to predict performance links/triplets (*solved* or *not-solved* links) in the KG between algorithm configurations and problem instances with a given target precision in a fixed-budget scenario. The KGs combine problem landscape and algorithm performance data with data related to the modular algorithm configuration.

The results indicate that our proposed classifier outperforms the baseline in balanced classification scenarios when performance triples are randomly selected for the test set (a classic KG completion scenario). However, in imbalanced classification scenarios, the classifier's performance decreases and falls below the baseline. In a more rigorous evaluation scenario, where we predict all performance links for problem instances and algorithm configurations that appear exclusively in the test set, we observed similar patterns. To address the performance drop in imbalanced classification, we modified the proposed pipeline and trained a Random Forest classifier on top of the learned embeddings, which improved the classifiers performance.

Our pipeline assumes a transductive learning setting, which limits its applicability to

unseen problems or algorithms that do not appear in the training graph. This limitation arises because transductive learning relies on the specific instances observed during training, meaning it cannot generalize to entirely new instances outside the training set. Consequently, while our approach is effective for predicting performance within the scope of the known graph structure, it cannot be directly applied to novel problems or algorithm configurations. To overcome this limitation, an inductive learning approach, particularly one that can handle regression tasks within a graph-based framework, is needed.

Chapter 8

Graph Neural Networks for Predicting Algorithm Performance in Numerical Black Box Optimization

Graph Neural Networks (GNNs) have emerged as a powerful tool for learning from relational data, effectively capturing the complex relationships between entities in the data. The BBO benchmarking data explored in this dissertation is most naturally represented using a relational structure, reflecting the intricate connections between algorithms, their parameters, and problem characteristics. Given the inherently relational nature of this data, GNNs offer a promising framework to explore for meta-learning tasks, such as algorithm performance prediction.

This chapter builds upon the work presented in Chapter 7, where a transductive learning setup and scoring-based knowledge graph reasoning were used for algorithm performance prediction. Here, we shift focus to exploring the potential of GNNs in an inductive setup, aiming to predict performance for problems not encountered during training. Additionally, instead of treating the task as a binary classification—predicting whether an algorithm reaches a specified performance precision within a given budget of function evaluations—we focus on predicting the exact performance value at that budget.

This chapter begins by defining the problem (Section **??**), highlighting the limitations of transductive approaches and the advantages of GNNs for analyzing relational data. We then present the methodology (Section 8.2, detailing the heterogeneous graph representation of the BBO benchmarking data, the architecture of the heterogeneous GNN models, and the training process. This is followed by a comprehensive description of the experimental setup 8.3. Next, the results and discussion section (Section 8.4) compares the proposed GNN-based approaches to baseline methods and evaluates model explainability using GNNExplainer to reveal key structural patterns that influence predictions. Finally, the chapter concludes with a summary of findings and insights **sec:summary_gnn**.

All data and code related to this chapter are publicly available on GitHub at: `https://github.com/KostovskaAna/GNN4PerformancePrediction`.

8.1 Problem Definition and Related Work

In the previous chapter, we addressed the task of performance prediction for modular optimization algorithms within a discretized framework. In this context, "discretized" means that the entire knowledge graph was treated as a discrete structure, and the task at hand was framed as a binary classification problem. Specifically, the focus was on

determining whether a given algorithm could solve a problem within a specified precision threshold using a fixed number of function evaluations. This was achieved using classical knowledge graph embedding (KGE) methods, such as the scoring-based ComplEx model. These methods treat the knowledge graph in a classical manner, assuming no additional node features exist and relying solely on relationships between entities. The resulting embeddings were inherently transductive, limiting generalization to only those nodes that appeared during training.

However, in real-world scenarios, it is often necessary to predict algorithm performance on previously unseen problems. This requires an inductive approach that generalizes beyond the training data to new nodes. To address this, we explore the use of Graph Neural Networks (GNNs), specifically message-passing architectures. Unlike traditional KGE methods, GNNs are capable of learning functions that generate node representations dynamically based on the graph structure and node features. This enables GNNs to generalize to unseen nodes, making them a suitable choice for inductive tasks [252].

While GNNs have not yet been explored for predicting the performance of modular optimization algorithms (to the best of our knowledge), they have been successfully applied to performance prediction tasks in other domains. For instance, GNNs have been used to predict the performance of neural network architectures by representing them as computation graphs. Lukasik et al. [253] proposed a surrogate model leveraging GNNs to predict the performance of unseen neural architectures, showcasing its effectiveness on the NAS-Bench-101 dataset [254]. Similarly, Singh et al. [255] developed a GNN-based performance model that represents deep neural networks (DNNs) as graphs, effectively capturing inter-stage interactions and predicting runtime performance with greater accuracy than traditional methods. Chai et al. [256] introduced PerfSAGE, a generalized inference performance predictor based on a Graph Neural Network (GNN). PerfSAGE predicts inference latency, energy consumption, and memory footprint for arbitrary deep neural networks deployed on edge devices.

In this chapter, we reformulate algorithm performance prediction as a node regression problem, aiming to predict continuous performance metrics (i.e., algorithm precision) for BBO problem instances. To accomplish this, we adapt the knowledge graph representation by incorporating node features, enabling GNNs to leverage both the relational and feature-based information. For example, optimization problem nodes are enriched with ELA features as their node attributes, rather than representing these features as separate nodes within the graph. It is important to emphasize that the graph structure, as in Chapter 7, remains grounded in the ontology terms developed in Chapter 4, providing a robust semantic foundation for the representation.

The primary objectives of this chapter are structured as follows:

- **Evaluating the applicability of the knowledge graph formalism in an inductive setup for algorithm performance prediction:** This chapter investigates whether knowledge graphs can be effectively utilized in an inductive learning context, addressing the limitations of the transductive approach discussed in Chapter 7. Additionally, we aim to determine if the proposed formalism can be adapted to **predict exact performance values** for algorithms, rather than being limited to binary classification tasks that evaluate whether an algorithm meets a specified performance threshold.

- **Assessing the performance of different GNN architectures:** We aim to compare two widely used GNN methods, GraphSage [13] and GAT [14], to evaluate their effectiveness in performance prediction and identify if there are any significant differences in their predictive capabilities.

- **Comparing GNN-based methods to traditional approaches:** Another objective is to compare the GNN-based approach to traditional machine learning models, specifically the Random Forest regressors explored in Chapter 6, and to evaluate whether GNNs can achieve superior predictive performance in this domain.

- **Investigating the explainability of GNN models:** As GNNs are often viewed as black-box models, we aim to explore whether graph explainability techniques can provide insights into their inner workings by identifying which relational information and node features are most important for predictions. This objective seeks to enhance the transparency and usability of GNN models in the context of algorithm performance prediction.

8.2 Methodology

In this section, we outline the methodology used in our study. This includes the graph representation of the BBO benchmarking data, the training process for heterogeneous GNNs, and the GNN architecture design, which describes the general architecture of our heterogeneous GNN models.

8.2.1 Graph representation

In this chapter, we represent the BBOB problems, modular optimization algorithms, their descriptors, and the corresponding algorithmic performance using a heterogeneous graph.

A *heterogeneous graph* (HG) [257] is defined as a graph $G = \{\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{T}\}$, where $\mathcal{V}$ represents the set of nodes, $\mathcal{E}$ represents the set of edges, $\mathcal{R}$ represents the set of relation types, and $\mathcal{T}$ represents the set of node types. Each node $v \in \mathcal{V}$ is associated with a node type through a mapping function $T(v) : \mathcal{V} \to \mathcal{T}$, and each edge $e \in \mathcal{E}$ is associated with a relation type through a mapping function $R(e) : \mathcal{E} \to \mathcal{R}$. For the graph to be considered heterogeneous, the sum of distinct node and relation types must be greater than two, i.e., $|\mathcal{T}| + |\mathcal{R}| > 2$.

In addition to type mappings, each node $v \in \mathcal{V}$ is associated with a feature vector $\mathbf{x}_v \in \mathbb{R}^d$, where d denotes the dimensionality of the feature space. These node features encode the properties of the nodes.

In Figure 8.1, we illustrate the meta-graph of the BBO heterogeneous graph that we propose. The meta-graph serves as a meta-template that abstracts the graph structure by representing node types as nodes and edge types as relationships between them. This schema provides a higher-level view of the heterogeneous graph, summarizing the relationships between different types of entities and their interactions.

Specifically, the meta-graph in Figure 8.1 illustrates the relationships between various components. The nodes in the graph are categorized into distinct types, such as parameter, parameter class, algorithm execution part, algorithm, performance, and BBOB problem, reflecting the different entities involved in our framework. The directed edges between these nodes represent specific relationships, such as has-parameter, has-parameter-class, controls-algorithm-execution-part, has-algorithm, and has-problem.

For instance, the has-parameter edge links an algorithm to its corresponding parameter, while the controls-algorithm-execution-part edge describes how a parameter class governs specific aspects of an algorithm's execution. Similarly, the has-algorithm edge connects performance metrics to the algorithm responsible for producing those results, and the has-problem edge indicates which BBOB problem is associated with a particular performance outcome.

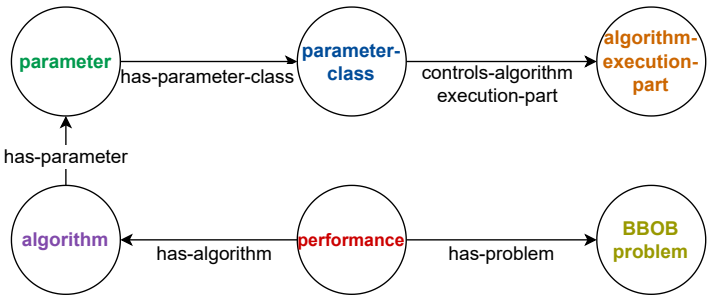


Figure 8.1: The meta-graph for the BBO heterogeneous graph. It consists of six node types and five edge types, representing the relationships between different components.

In Figure 8.2, we illustrate a concrete instantiation of the meta-graph, providing a snapshot of the corresponding heterogeneous graph. This example illustrates how a subset of the graph is structured for the modCMA algorithm variants. The example includes two connected BBOB problems, two algorithm variants connected to four performance nodes, and the parameters associated with each algorithm variant. These parameters belong to the local_restart, base_sampler, and elitism parameter classes. The local_restart and base_sampler parameter classes influence the mutation part of the algorithm execution, while elitism impacts the selection process. In this chapter, one graph is defined for each unique combination of problem dimensionality, runtime budget, and modular algorithm, resulting in a collection of distinct graphs for the different experimental setups.

8.2.2 Training heterogeneous GNNs

In Section 2.6.2, we have introduced the general concept of GNNs and their message-passing mechanisms, which has initially been designed for homogeneous graphs. However, our graph data is heterogeneous, thus introducing additional complexity due to the presence of multiple node and relation types. This added complexity must be addressed appropriately when training a message-passing GNN, to ensure an effective learning of node representations. To accommodate the heterogeneous structure of the BBO graph, we apply an approach designed for computing relation-specific convolutions to each relation type in the graphs.

In a heterogeneous graph $G = \{\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{T}\}$, each relation type $r \in \mathcal{R}$ governs the message passing between nodes of different types. Specifically, for a relation $r \in \mathcal{R}$, where source nodes belong to type $T(u) = t_u$ and destination nodes belong to type $T(v) = t_v$, the message from node u to node v at layer l is computed as:

$$m_{r,u\to v}^{(l)} = \text{MESSAGE}_r^{(l)}\left(\mathbf{h}_u^{(l-1)}\right), \quad u \in \mathcal{N}_r(v)$$

Here, $\mathcal{N}_r(v)$ represents the set of neighbors of node v connected via relation r, and $\mathbf{h}_u^{(l-1)}$ is the hidden representation of the source node u from the previous layer. The message function $\text{MESSAGE}_r^{(l)}$ is specific to a relation r and can vary depending on the chosen GNN method. For example, $\text{MESSAGE}_r^{(l)}$ might be a simple linear transformation of node features, or it could involve more sophisticated mechanisms, such as attention-based methods, where the importance of neighboring nodes is dynamically adjusted based on their features.

Once the messages are computed, the next step involves aggregation of these messages

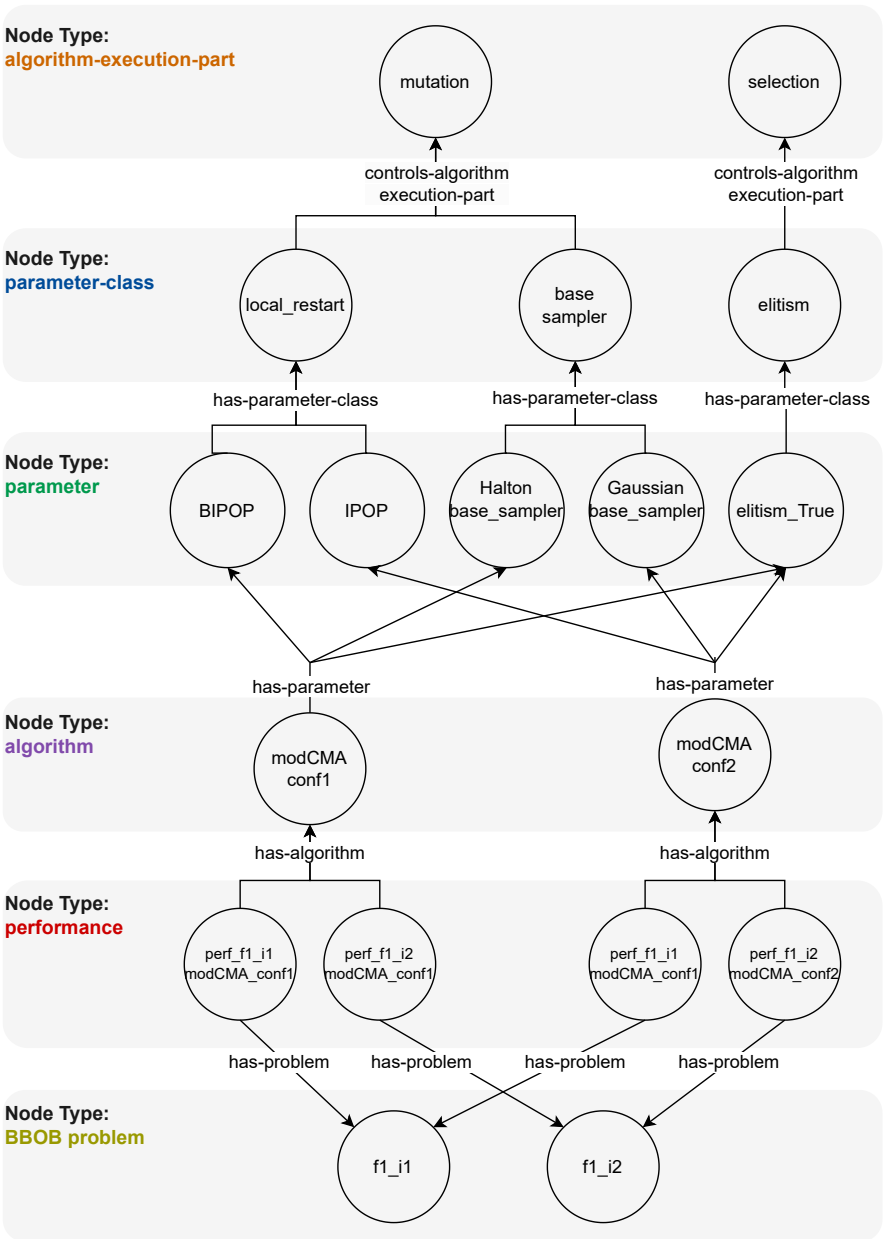


Figure 8.2: An illustration of an instantiation of the meta-graph, showing a snapshot of the BBO heterogeneous graph defined for a specific combination of problem dimensionality, runtime budget, and modular algorithm class.

per relation type. For each relation r, the messages passed from the neighbors of v are aggregated into a single representation as follows:

$$\mathbf{h}_{r,v}^{(l)} = \text{AGGREGATE\_PER\_RELATION}_r^{(l)} \left(\{ m_{r,u \to v}^{(l)} : u \in \mathcal{N}_r(v) \} \right)$$

This aggregation function, $\text{AGGREGATE\_PER\_RELATION}_r^{(l)}$, can vary depending on the application. It may be instantiated as a simple mean, sum, max pooling function or concatenation of the incoming messages, as well as a more complex function, such as those found in attention mechanisms [14] or transformer architectures [258].

Finally, since each node v may participate in multiple relations, the representations obtained from the per-relation aggregations are further aggregated across relations. This cross-relation aggregation combines information from all relations that involve node v, leading to the updated representation for v at layer l:

$$\mathbf{h}_v^{(l)} = \text{AGGREGATE\_CROSS\_RELATION}^{(l)} \left(\{ \mathbf{h}_{r,v}^{(l)} : r \in R(\mathcal{E}_v) \} \right)$$

where, $R(\mathcal{E}_v)$ is the set of all relations in which node v is involved. The cross-relation aggregation function $\text{AGGREGATE\_CROSS\_RELATION}^{(l)}$ aggregates the results from different relation-specific convolutions. This is done using a specified method, such as summing or averaging the outputs from the various relation modules. The aggregation is performed per destination node type, meaning that nodes of the same type can receive updates from multiple relations, and these updates need to be combined. After the cross-relation aggregation, the final hidden representation of node v is typically updated by applying a non-linear transformation.

8.2.3 GNN architecture design

In this study, we implement the proposed framework for training heterogeneous GNNs, as outlined earlier. The model architecture consists of multiple stacked GNN layers, where each layer is tailored to the specific GNN algorithm employed. This is followed by aggregation across relation types to capture the heterogeneity of the graph. This flexible design enables experimentation with different GNN algorithms, such as GraphSAGE [13] and GAT [14]. Relation-specific convolutions are first applied independently to each relation type using the selected GNN algorithm. The resulting features are then aggregated across all relations to form the final node representations. The overall architecture is depicted in Figure 8.3.

Each GNN layer in the architecture performs message passing and relation-specific aggregation, followed by a cross-relation aggregation step. Non-linearity is introduced through an activation function, and dropout is applied after each convolutional block to prevent overfitting.

The primary task in this study is node regression, where the objective is to predict algorithm performance on various problem instances. Each performance node is associated with a numerical score that reflects the performance of a particular algorithm on a specific BBOB problem instance. The model learns to predict these scores by leveraging the relationships captured in the heterogeneous graph and learning node embeddings. The output from the GNN layers is passed through a fully connected linear layer to predict the performance score ($\hat{y}$) for the performance nodes.

Note that, such trained model is capable of generalizing across different algorithm variants, predicting the performance of *all modular algorithm variants* for a given runtime budget and problem dimensionality. Also, although the input graph shown in Figure 8.2 is directed, with edges representing the flow of information from one node type to another (e.g., from algorithm to performance or from parameter class to algorithm execution

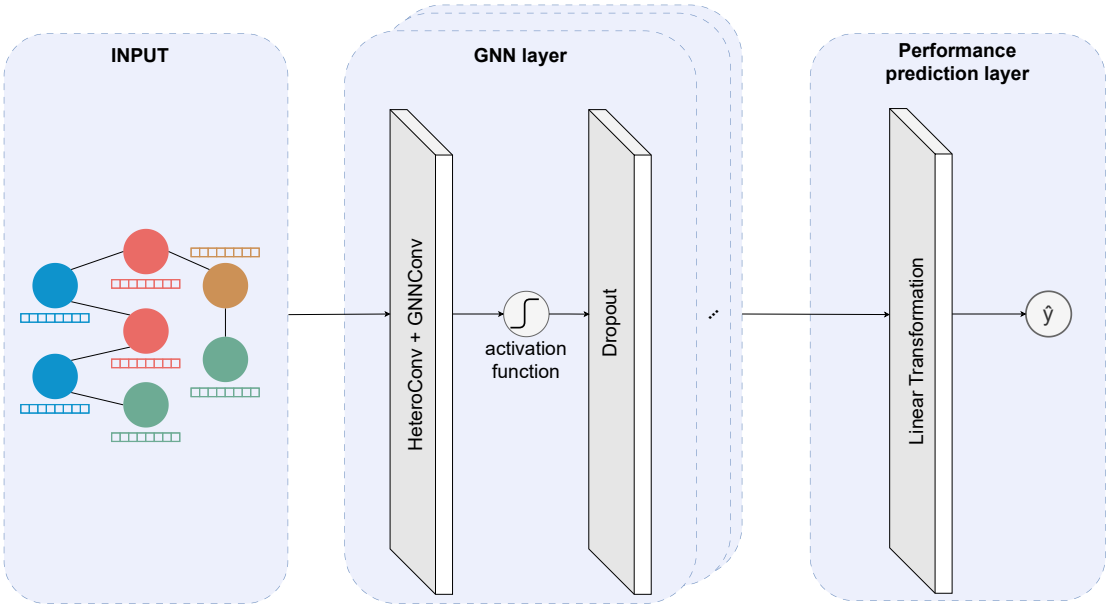


Figure 8.3: An overview of the general GNN architecture for predicting algorithm performance using heterogeneous graphs.

part), we incorporate reverse edges during message passing. This modification effectively transforms the graph into an undirected structure, allowing bidirectional information flow between connected nodes and enhancing the representation learning process.

8.3 Experimental Setup

We use the Deep Graph Library (DGL) [259] to implement Graph Neural Networks (GNNs) for modular optimization algorithm performance prediction. Specifically, we employ the `HeteroConv` layer for cross-relation aggregation, alongside the `SageConv` and `GATConv` layers, which implement the GraphSAGE and GAT algorithms, respectively. For a comprehensive explanation of these algorithms, we direct readers to the works of Hamilton et al. [13] for GraphSAGE and Veličković et al. [14] for GAT.

Within each relation, message aggregation is performed using the mean function, while aggregation across different relations is carried out using summation. The feature vectors for the BBOB problem nodes are based on 46 Exploratory Landscape Analysis (ELA) features, as introduced in Chapter 4. In contrast, the feature vectors for the remaining nodes are initialized randomly using the Kaiming uniform distribution [260]. After each GNN layer, we apply the GELU activation function [261] to introduce non-linearity.

We use a 4-hop neighborhood during learning to ensure that all relational information about the algorithms, which are located up to four hops away from the performance nodes, is effectively incorporated.

Additionally, a nested cross-validation scheme is used to tune the dropout rate (0.1, 0.2, 0.3) and the dimensionality of the learned embeddings (32, 64, 128). For the GAT models, an additional parameter, the number of attention heads (4, 8), is also tuned. All experiments are repeated 10 times to account for randomness in initialization and to provide more reliable results. The model is trained using the L1 loss function, defined as:

$$\text{L1Loss} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

where N is the number of performance nodes. The model performance is evaluated using two metrics: R^2 and mean squared error (MSE).

To ensure robustness, we employ a *leave-instance-out nested cross-validation* procedure, as outlined in Chapter 6. The experimental setup in this case remains the same as in Chapter 6. To summarize, we have 324 variants of the modCMA algorithm, 567 variants of modDE, 6 runtime budgets, 2 problem dimensionalities, and the first five instances of the 24 BBOB problems. Each combination of these parameters defines a distinct learning task, with an associated graph.

The model is optimized using the Adam optimizer [250], and a ReduceLROnPlateau scheduler is employed. We start with an initial learning rate of 0.1, which is reduced by a factor of 0.5 every 20 epochs if the validation performance plateaus. The model is trained for a total of 200 epochs.

8.4 Results and Discussion

Building on the methodology and experimental setup outlined above, we conducted computational experiments to predict the performance of modular algorithms. The results of the predictive models evaluation are presented in Table 8.1 and Table 8.2, which show the R^2 scores and MSE scores, respectively. Each table provides results for two problem dimensionalities and six evaluation budgets, comparing the performance of GraphSAGE and GAT. Additionally, we include the results of RF regression models presented in Chapter 6 as a baseline for comparison with the GNN models.

Table 8.1: The R^2 scores of the GraphSage, GAT and RF regression models for predicting the performance of CMA-ES and DE algorithm variants for the BBOB problem instances in 5 and 30 dimensions where the target precision is measured at $B \in \{50D, 100D, 300D, 500D, 1000D, 1500D\}$ function evaluations.

Budget	CMA-ES - 5D			CMA-ES - 30D		
	GraphSage	GAT	RF	GraphSage	GAT	RF
50D	**0.80**	0.77	0.76	**0.94**	0.93	**0.94**
100D	**0.78**	0.73	0.77	**0.94**	0.93	0.92
300D	**0.61**	0.58	**0.61**	**0.88**	0.84	0.85
500D	**0.70**	0.65	**0.70**	**0.88**	0.84	0.83
1000D	**0.73**	0.69	**0.73**	**0.87**	0.81	0.81
1500D	0.71	0.67	**0.73**	**0.89**	0.84	0.84
	DE - 5D			DE - 30D		
	GraphSage	GAT	RF	GraphSage	GAT	RF
50D	**0.90**	0.86	0.88	**0.96**	0.92	0.94
100D	**0.91**	0.88	0.88	**0.96**	0.94	0.94
300D	**0.90**	0.85	0.86	**0.95**	0.92	0.94
500D	**0.86**	0.83	0.84	**0.95**	0.93	0.94
1000D	0.75	0.74	**0.78**	**0.94**	0.90	0.92
1500D	0.76	0.71	**0.78**	**0.93**	0.90	0.92

In both cases (R^2 and MSE), GraphSAGE generally outperforms the RF models across dimensionalities and budgets. Interestingly, GAT exhibits lower performance compared to both GraphSAGE and RF. This suggests that the choice of GNN architecture significantly influences the performance of predictive models. While GATs are generally considered

Table 8.2: The MSE scores of the GraphSage, GAT and RF regression models for predicting the performance of CMA-ES and DE algorithm variants for the BBOB problem instances in 5 and 30 dimensions where the target precision is measured at $B \in \{50D, 100D, 300D, 500D, 1000D, 1500D\}$ function evaluations.

Budget	CMA-ES - $5D$			CMA-ES - $30D$		
	GraphSage	GAT	RF	GraphSage	GAT	RF
$50D$	**0.75**	0.95	0.78	**0.15**	0.19	**0.15**
$100D$	**1.16**	1.41	1.22	**0.19**	0.25	0.27
$300D$	**3.68**	3.94	3.98	**0.75**	0.99	1.03
$500D$	**4.39**	5.13	4.85	**0.85**	1.12	1.27
$1000D$	**4.38**	4.99	5.22	**1.09**	1.63	1.72
$1500D$	**4.57**	5.34	5.19	**1.34**	2.07	1.88
	DE - $5D$			DE - $30D$		
	GraphSage	GAT	RF	GraphSage	GAT	RF
$50D$	**0.36**	0.50	0.37	**0.21**	0.31	0.26
$100D$	**0.39**	0.51	0.43	**0.19**	0.29	0.25
$300D$	**0.62**	0.92	0.81	**0.27**	0.44	0.30
$500D$	**1.00**	1.21	1.04	**0.32**	0.47	0.38
$1000D$	2.08	2.16	**1.95**	**0.49**	0.81	0.54
$1500D$	**2.26**	2.56	2.34	**0.58**	0.85	0.63

more expressive due to their attention mechanism, this additional complexity appears to lead to overfitting and reduced generalizability on the test set.

Although GraphSAGE achieves slightly better performance than RF, the improvement is not substantial. For instance, in the best case for R^2 scores (predicting CMA-ES on 30D problems with a 1000D budget), GraphSAGE improves from 0.81 to 0.87, representing a relative improvement of approximately **7.41%**. This improvement highlights that while GNN models can offer advantages, their benefits may depend on the specific evaluation settings.

8.4.1 The impact of the GNN receptive field

The results presented above are based on architectures composed of 4 GNN layers, allowing the model to aggregate information from nodes up to 4 hops away from the performance nodes. We chose 4 layers to ensure all relational data is incorporated. However, in this section, we investigate the influence of the number of GNN layers on the final results. Using the same experimental setup, we vary the number of layers from 1 to 4, focusing on GraphSAGE models due to their superior performance compared to GAT.

When employing an architecture with one GNN layer, the model can only aggregate information from nodes directly connected to a performance node (i.e., 1-hop neighbors). In our case, this includes the BBOB problem and the algorithm associated with the performance node, along with the ELA features of the problem as node features. Consequently, the model cannot leverage information about the algorithm configuration and its parameters.

As the number of layers increases, the receptive field of the GNN model expands. With 2 GNN layers, the model gains access to information about the algorithm parameters and due to the addition of the reverse edges it can access the performance nodes connected with the same problem instance. With 3 GNN layers, among other information, it incorporates information about the parameter classes to which these parameters belong. With 4 GNN layers, the model additionally learns from information about the parts of the algorithm execution controlled by each parameter class.

Figures 8.4 and 8.5 illustrate the R^2 performance of the GraphSAGE models with

varying numbers of layers (from 1 to 4), across different performance dimensionalities and budgets, for CMA-ES and DE, respectively.

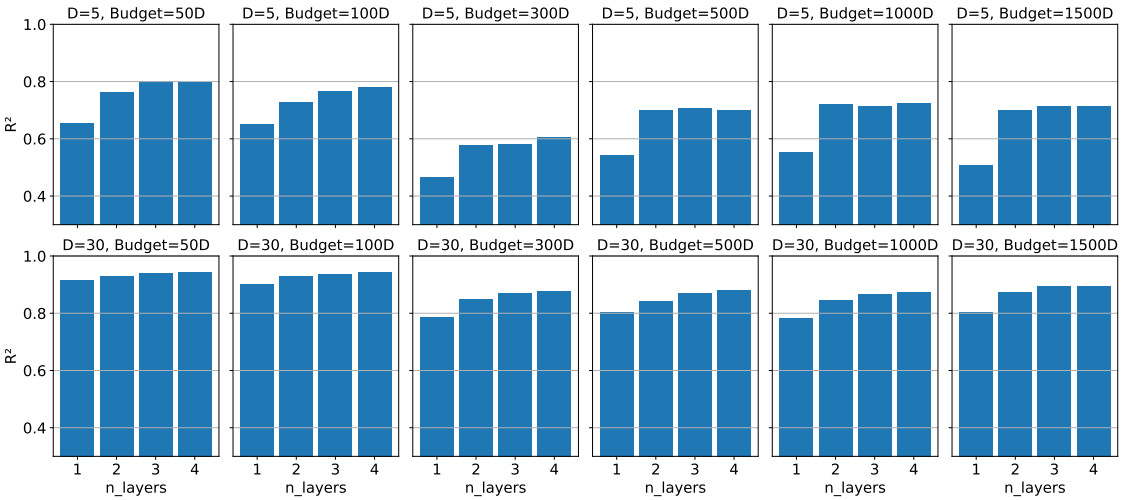


Figure 8.4: R^2 performance of GraphSAGE models with 1 to 4 layers for CMA-ES across different dimensionalities and budgets.

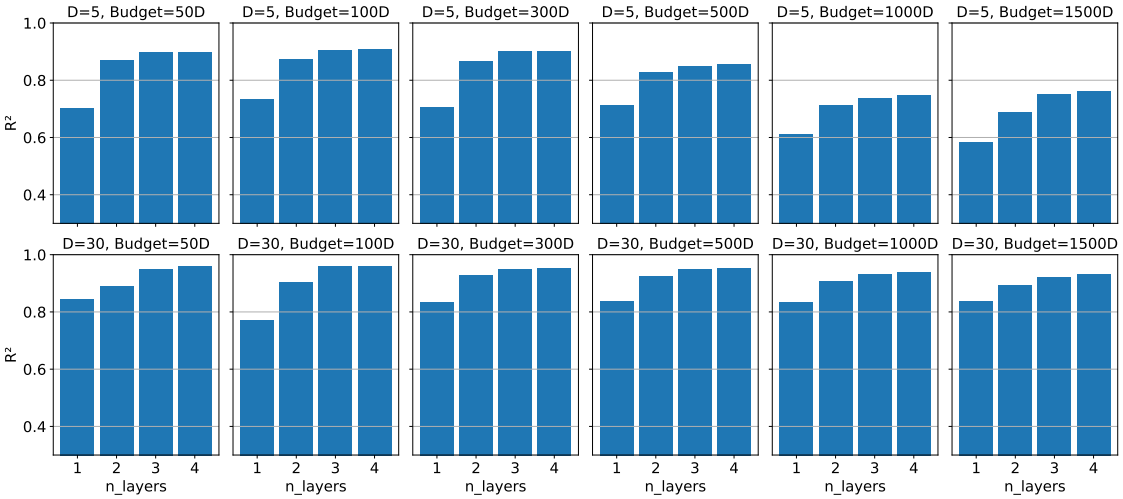


Figure 8.5: R^2 performance of GraphSAGE models with 1 to 4 layers for DE across different dimensionalities and budgets.

We observe that increasing the number of GNN layers generally improves R^2 predictive performance, indicating that the model effectively utilizes relational information as its receptive field expands. Notably, the most significant performance improvement occurs when increasing the number of layers from 1 to 2. This can be attributed to the incorporation of algorithm parameter information at the 2-hop neighborhood, which provides critical insights about the algorithms. In contrast, the smallest improvement is observed when increasing the number of layers from 3 to 4. This can be explained by the fact that the information about the part of the algorithm execution controlled by specific parameters has minimal impact on R^2 performance.

8.4.2 Explaining GNN predictions

Explainability in GNNs is essential for interpreting the model's decision-making process, particularly in complex applications such as predicting algorithm performance in numerical black-box optimization. By understanding which parts of the graph structure and which node features drive the model's predictions, we gain valuable insights into the relationships captured by the GNN. For heterogeneous GNNs, explainability becomes even more important, as it allows us to analyze the contributions of different node and relation types, clarifying how diverse entities and interactions affect the predictions.

In this study, we use the GNNExplainer algorithm, originally proposed in GNNExplainer: Generating Explanations for Graph Neural Networks [15], and adapted for heterogeneous graphs. GNNExplainer identifies small, important parts of the graph structure and specific features of nodes that contribute most to the GNN model's predictions.

To create these explanations, the explainer model learns edge masks M and feature masks F, optimizing the following objective:

$$l(y, \hat{y}) + \sum_{r \in \mathrm{R}} (\alpha_{1,r} \|M_r\|_1 + \alpha_{2,r} H(M_r)) + \sum_{t \in \mathrm{T}} (\beta_{1,t} \|F_t\|_1 + \beta_{2,t} H(F_t))$$

where $l(y, \hat{y})$ measures the loss between the original prediction y and the masked prediction $\hat{y}$, ensuring prediction quality. The terms $\alpha_{1,r}\|M_r\|_1$ and $\beta_{1,t}\|F_t\|_1$ impose L_1-norm regularization on edge masks M_r and feature masks F_t, respectively, to enforce sparsity. The entropy terms $\alpha_{2,r}H(M_r)$ and $\beta_{2,t}H(F_t)$ reduce uncertainty in the masks, guiding them to focus on the most relevant edges and features.

The explainer model generates explanations for each performance node individually. This approach enables explainability at a local level, allowing us to analyze the model's behavior specific to each node. In this section, we demonstrate how this explainability technique can be applied for local explanations. However, to achieve global explainability, future work could focus on aggregating the edge and node feature masks across all performance nodes. Such aggregation would capture general patterns and identifying key relationships that consistently influence the model's predictions.

To evaluate local explainability, we focus on interpreting the predictions made by a GraphSAGE model for a randomly selected performance node, which predicts the performance of a CMA-ES algorithm variant in the $5D$ problem setting with a $100D$ budget. While this analysis highlights GraphSAGE, it is important to emphasize that GNNExplainer is model-agnostic and can be applied to various GNN architectures.

We use the implementation with default hyperparameters, including the masking terms $\alpha_1 = 0.005$, $\alpha_2 = 1.0$, $\beta_1 = 1.0$, and $\beta_2 = 0.1$, with mean squared error (MSE) as the loss function.

We analyze the learned node feature mask for the problem nodes, which can be interpreted as the importance of the ELA features. Only the problem node features are assessed, as the input features for other node types are randomly generated and, therefore, not meaningful for interpretation. Figure 8.6 illustrates the top 15 most important ELA features based on the learned node feature mask. The analysis of ELA feature importance highlights that features such as lin_w_interact.adj_r2, diff_median_25, and ratio_median_25 are among the most influential in predicting performance. This indicates that these features capture critical characteristics of the problem landscape, providing insights for the model predictions.

We also analyze the edge mask by calculating the mean values for each edge type. This analysis provides insights into the relative importance of different edge types within the graph. Figure 8.7 illustrates the importance of each edge type based on the averaged edge mask values.

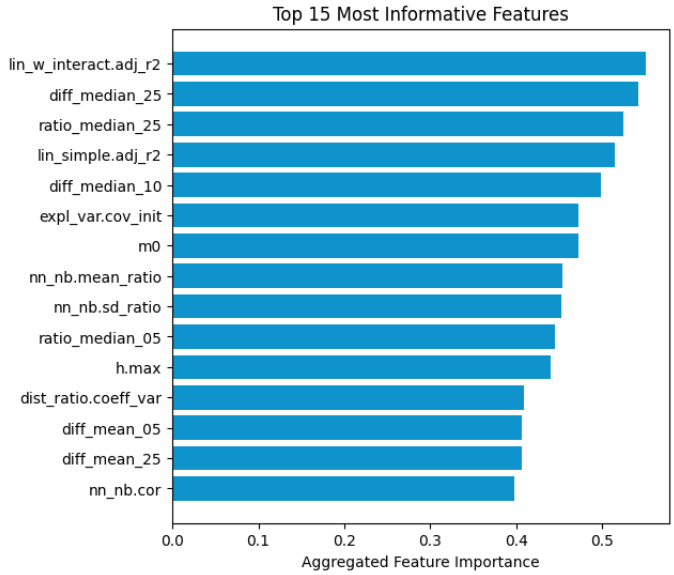


Figure 8.6: The top 15 most important ELA features for explaining the predictions of a performance node.

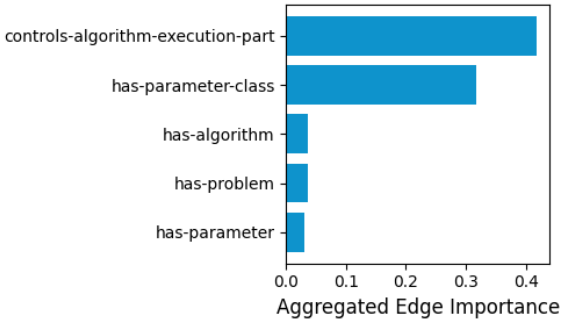


Figure 8.7: Aggregated edge importance scores for the different edge types in the graph.

From the results, we observe that the hasalgorithm, hasproblem, and hasparameter edge types exhibit lower importance compared to controls-algorithmexecutionpart and hasparameterclass. This difference can be explained by the bidirectional nature of edges in our graph. At a 4hop neighborhood from the central performance node, the graph includes other algorithm nodes and problems that are not directly connected to the performance node being predicted. Since we average the importance values across all instances of a given edge type, the lower scores for these three edge types suggest that the influence of other algorithms and problems in the graph is relatively small.

However, if we consider only the has-problem edge type associated with the immediate problem node connected to the performance node, its importance is high. When this importance is averaged with contributions from other problem nodes in the local neighborhood, the overall score becomes lower. This indicates that while the direct connections are critical for predictions, the influence of more distant nodes diminishes, leading to reduced aggregated importance for these edge types.

8.5 Summary

In this chapter, we investigated the use of GNNs for predicting modular optimization algorithm performance in numerical BBO. Addressing the limitations of transductive learning approaches discussed in Chapter 7, we introduced an inductive learning framework using heterogeneous GNNs. This approach enables generalization to previously unseen BBO problem instances, overcoming the constraints of traditional KGE methods that are limited to entities and relationships observed during training. Furthermore, the transition from binary classification to node regression allowed for the prediction of exact performance values, providing a more detailed understanding of algorithm behavior.

The chapter demonstrated the applicability of KGs in an inductive setup by incorporating both node features and relational structures. This approach proved effective for predicting the performance of modular algorithms across various experimental setups. By comparing GraphSAGE and GAT architectures, we found that GraphSAGE consistently outperformed GAT. This analysis highlighted the significance of selecting appropriate GNN architectures for performance prediction tasks.

In addition, the performance of the GNN-based framework was compared to traditional machine learning models, such as Random Forest regressors. While the GNN models offered slight improvements in predictive performance scores, their ability to generalize to unseen problem instances was a key advantage. This inductive capability in relational learning scenarios makes GNNs a suitable choice for real-world applications where data extends beyond the training set. The analysis of the GNN receptive field revealed that increasing the number of layers improves the model's ability to learn from broader relational contexts.

To address the inherent "black-box" nature of GNNs, we employed GNNExplainer to analyze which parts of the graph structure and node features contributed most to the model's predictions. By focusing on local-level explainability, we demonstrated how the technique provides insights into individual predictions. Future work could explore aggregating these explanations across performance nodes to achieve global explainability.

In conclusion, this chapter showcased the potential of heterogeneous GNNs for leveraging relational and feature-based information to predict algorithm performance. The findings lay the groundwork for future advancements in the utility of GNN-based methods, paving the way for more effective frameworks for performance prediction in single-objective numerical BBO.

Chapter 9

Conclusions

This chapter highlights the scientific contributions of this dissertation, discussing how the the research questions were addressed and providing an outline of potential directions for future work.

9.1 Research Outcomes and Scientific Impact

In this section, we revisit the research questions posed in the introduction of the dissertation and summarize the key findings of the conducted research. Each question is addressed by summarizing the main insights and contribution, and highlighting their broader impact on the field.

R1. Can an MLC ontology-based semantic annotation scheme be designed and applied to annotate MLC benchmarking data to enable easy data accessibility, improved querying capabilities, increased reusability, and support for automated data integration and domain knowledge sharing?

This research question is addressed in Chapter 3, where an MLC ontology-based semantic annotation scheme was developed, representing a primary outcome of this work. The scheme was applied to annotate MLC benchmark data, resulting in the creation of the MLCBench online catalog, which showcases the practical application of the designed scheme. All annotated data is easily accessible via a SPARQL endpoint, enabling users familiar with semantic technologies to perform diverse and complex queries efficiently. Additionally, the data is accessible through the MLCBench graphical user interface, ensuring usability for a broader audience. The semantic annotations make the data machine-readable, facilitating automated data integration and promoting domain knowledge sharing.

R2. Can a BBO benchmarking ontology be designed and applied to annotate BBO benchmarking data to enhance data accessibility, querying capabilities, reusability, and enable automated data integration and domain knowledge sharing?

This research question is addressed in Chapter 4, where a novel ontology named OPTION was developed to formally represent the benchmarking domain of BBO. The ontology was applied to annotate a substantial pool of benchmarking data originating from multiple platforms, resulting in the creation of the OPTION knowledge

base (KB). These platforms utilized distinct storage formats, demonstrating the versatility of the ontology in integrating diverse data sources. The annotated data is made easily accessible via a REST API, facilitating seamless integration with external tools and workflows. Specifically, this API is incorporated within the IOHprofiler environment, further showcasing the practical utility and efficient querying capabilities of the OPTION KB. The semantic annotations produced using OPTION ensure machine-readability, enabling increased reusability of the benchmark data across diverse applications. Finally, these annotations facilitate automated data integration and promote domain knowledge sharing by providing a standardized and interoperable representation of BBO benchmarking data.

R3. Does the development of a data-driven AAS pipeline for multi-label classification (MLC) lead to better AS practices by leveraging dataset-specific characteristics, and how does it compare to static approaches using a single algorithm across all datasets?

We addressed this research question in Chapter 5, where we developed and validated a data-driven automated algorithm selection pipeline for MLC. By leveraging dataset-specific characteristics tailored to MLC datasets, we demonstrated that predictive models built using these features consistently outperform static approaches relying on a single algorithm performing best on average. This result was confirmed across multiple machine learning approaches and performance evaluation metrics. The proposed pipeline has the potential to significantly improve algorithm selection practices, enabling more efficient and tailored solutions for diverse real-world MLC applications.

R4. Can a systematic empirical analysis of modular BBO algorithm behavior, combined with algorithm and problem characterization, improve our understanding of the impact of algorithm modules and problem landscape characteristics on algorithm performance?

This question is studied in Chapter 6, where we developed an empirical pipeline to systematically evaluate modular BBO frameworks, such as modCMA-ES and modDE. By leveraging their modular structures, we analyzed the influence of individual modules on algorithm performance. Notably, the elitism module in modCMA-ES and the linear population size reduction module in modDE demonstrated substantial impacts on performance. Additionally, we leveraged problem landscape characteristics to train regression models for performance prediction. This analysis revealed consistent trends in the importance of specific landscape features across various configurations. These findings confirm that our systematic evaluation approach enhances the understanding of how algorithm modules influence performance, while demonstrating the predictive value of problem features for performance prediction. Furthermore, this work provides insights for designing more effective modular BBO algorithms, potentially enabling improved optimization performance across diverse problem landscapes.

R5. Are KGs as semantic data representations effective for predicting the performance of modular BBO algorithms?

This research question is explored in Chapters 7 and 8. In Chapter 7, we employ shallow scoring-based knowledge graph embedding models, such as ComplEx, in a transductive setup. These models predict whether modular algorithms like modCMA-ES and modDE solve specific problems within fixed budgets and precision thresholds. The results show that our approach surpasses baseline methods, particularly in balanced classification scenarios. Chapter 8 extends this by demonstrating the predictive power of Graph Neural Networks in an inductive setup. Together, these findings highlight the predictive power and versatility of KGs across diverse performance prediction tasks. This work underscores the potential of KGs as a foundational tool for integrating and exploiting semantic data, paving the way for more robust performance prediction frameworks in optimization and algorithm selection.

9.2 Final Conclusions and Future Work

Benchmarking data is a cornerstone of empirical research in ML and BBO, offering critical insights into algorithm behavior and performance trends. However, the true potential of benchmarking data lies in its effective representation and exploitation. Accurate and consistent representation ensures accessibility, interoperability, and reusability, while thoughtful exploitation reveals meaningful patterns, predicts performance, and supports informed decision-making in algorithm selection and design. Together, these aspects bridge the gap between raw data and actionable knowledge, advancing the state of the art in both ML and BBO. This dissertation demonstrates how formal semantic representation and innovative methodologies for benchmarking data exploitation can advance the state of the art in ML and BBO, paving the way for more robust, interpretable, and reliable research.

For the representation of benchmarking data, this dissertation has developed formal semantic models, including the semantic annotation schema for MLC benchmarking data and the OPTION ontology. These models are complemented by practical implementations: MLCBench, a semantic catalog for exploring and accessing MLC benchmarking data, and the OPTION Knowledge Base (KB), which enables access to BBO benchmarking data from different benchmarking platforms. While these contributions represent a significant progress in the representation of benchmarking data, several opportunities for future work remain.

One promising direction is the expansion of MLCBench through community contributions. Opening the knowledge base to inputs from the broader research community, supported by a curation layer to maintain high data quality, could enrich the repository and foster collaborative growth. Additionally, extending the schema to incorporate custom evaluation scenarios and more detailed representations of MLC algorithms, such as their hyperparameters, assumptions, constraints, and computational complexity, would further enhance the catalog's utility.

For the OPTION ontology, future efforts could focus on extending annotations to cover benchmarking data from additional platforms actively used in the optimization community, such as DEAP (Distributed Evolutionary Algorithms in Python) [262] and HyperBench [263]. This expansion would strengthen the ontology's role as a unified model for connecting diverse data formats and enabling full interoperability.

Another exciting avenue to explore involves the use of Description Logic (DL) within the OPTION ontology to classify algorithms and problems based on their characteristics. For example, high-level problem landscape properties such as ruggedness, or multimodality could be encoded with DL rules, thus enabling automated classification of problems using an ontology reasoner. Similarly, algorithms could be categorized based on their components, operators, or performance characteristics. These rules would form the basis for

a dynamic and adaptive algorithm taxonomy within the ontology. This approach would also support the seamless addition of new algorithms and problems to the ontology. By answering a set of predefined questions, such as specifying the high-level properties of a problem or the components of an algorithm, users could integrate new entities into the ontology without extensive manual intervention. The ontology reasoner would then apply the DL rules to automatically classify the new entries, ensuring that the ontology grows organically while maintaining consistency and coherence.

While we have developed ontology-based KBs, such as the MLCBench catalogue and the OPTION KB, the primary value of this work lies in the creation of the semantic annotation schema for MLC benchmarking data and the OPTION ontology. Both the MLC schema and the OPTION ontology establish a robust foundation for the formal semantic representation of benchmarking data, empowering the creation of decentralized knowledge bases. Researchers and developers are not constrained to contributing to centralized repositories; instead, they can use the schema and ontology to independently build and maintain their own KBs. Any KB developed using these standardized vocabularies will seamlessly integrate with others adhering to the same approach, enabling data sharing and interoperability across platforms and studies.

In situations where managing semantic triple stores of annotated data is not feasible, semantic web technologies offer an alternative solution through ontology-based data access (OBDA). OBDA facilitates the integration of heterogeneous data sources by mapping them onto a unified, queryable format without the need to change the storage of data in a semantic triple store or a KB. Instead, existing databases, regardless of their structure or format, can be aligned with semantic data models through the use of standardized vocabularies that they provide. This approach ensures that the advantages of semantic interoperability and consistency are preserved, even in contexts where traditional KBs are not suitable.

Building on the foundation of benchmarking data representation, this dissertation has focused on exploiting benchmarking data through data-driven methodologies for automated algorithm selection, analysis of algorithm behavior, and performance prediction.

In the MLC domain, we have demonstrated the potential of benchmarking data to enable data-driven decision-making in AAS. By assessing the predictive power of dataset meta-features and leveraging machine learning models, we showcased how the unique characteristics of MLC datasets can effectively guide the algorithm selection process. This approach addresses the inherent variability in MLC datasets, where no single algorithm consistently excels across all datasets. By tailoring algorithm selection to dataset-specific attributes, it ensures more efficient and effective utilization of computational resources.

Future work can build on our findings and contributions to MLC AS following several avenues. Expanding the diversity of MLC datasets in the AS pipeline would improve the generalizability and robustness of the findings, enabling models to better capture the variability in MLC tasks. Additionally, techniques for selecting representative dataset portfolios, such as those described in by Eftimov et al. [264] and Cenikj et al. [265], could be explored to mitigate bias by ensuring that AS models are not disproportionately influenced by certain types of dataset distributions or landscapes. Furthermore, while Random Forests proved effective in this study, future research could investigate alternative ML methods such as deep neural networks [41], k-nearest neighbors [266], support vector machines [267], or gradient boosting machines [268] to determine whether they offer improved performance or novel insights.

In the BBO domain, leveraging benchmarking data about modular black-box optimization algorithms, we developed methodologies to assess the influence of individual algorithm modules on overall performance. This general framework provides a robust approach to

understanding how specific components contribute to the behavior of modular optimization algorithms.

Future work could extend this methodology by applying it to other modular frameworks beyond modCMA-ES and modDE, thereby exploring its generalizability and adaptability across diverse optimization paradigms. Additionally, a promising avenue for research involves investigating the interplay and interactions between multiple algorithm modules. Understanding how combinations of modules influence performance could uncover synergistic configurations, enabling the design of more effective and efficient modular optimization algorithms.

Finally, this dissertation explores the use of KGs to represent benchmarking data. We argue that KGs are a suitable representation for predictive modeling studies involving such data. Unlike traditional feature-based tabular representations, KGs provide a structured yet flexible framework that captures the rich semantics and relationships inherent in benchmarking data, offering a refreshing perspective on the task. This novel approach lays the groundwork for innovative methodologies that could significantly enhance the utility of benchmarking data in ML and optimization research.

So far, we have explored the predictive power of KGs primarily with modular optimization algorithms. Extending this approach to other modular frameworks would be an interesting direction for future research. However, applying it beyond modular frameworks presents a more significant challenge. The methodology relies on representations for both algorithms and problems. While problem representations are well-established, structured and standardized representations for algorithms are often lacking.

To apply this approach effectively to non-modular algorithms, a formal and standardized vocabulary is needed to represent algorithm operators, their hyperparameters, and their interactions. Developing these representations is a complex and resource-intensive task that requires consensus and collaboration across the research community. We hope this work inspires efforts toward creating standardized, unified representations for black-box algorithms, facilitating the broader application of this methodology to diverse black-box optimization algorithms and expanding its impact. Furthermore, the general nature of this approach opens the door for its exploration in the domain of ML, where it could provide valuable insights into algorithm selection and performance prediction.

Future research could build on this foundation by incorporating continuous learning paradigms with Graph Neural Networks (GNNs) [101], enabling models to dynamically adapt as new benchmarking data becomes available. This approach aligns well with the iterative nature of empirical research, where new data and algorithms are continually added to the ecosystem. Additionally, integrating the KG-based approach with foundation models [269] could unlock new capabilities, allowing pretrained models to generalize across domains and tasks using the rich relational structure of KGs as a grounding mechanism.

Using transfer learning also presents a promising avenue to follow, where learned representations from one set of benchmarking studies could be applied to other domains or tasks, enhancing both efficiency and performance in scenarios with limited labeled data. Furthermore, the interplay of GNNs and transfer learning in KG representations [270] offers a compelling strategy for building robust models that can effectively leverage diverse datasets without extensive retraining.

References

[1] N. Sharma, R. Sharma, and N. Jindal, "Machine learning and deep learning applications - a vision," *Global Transitions Proceedings*, vol. 2, no. 1, pp. 24–28, 2021, 1st International Conference on Advances in Information, Computing and Trends in Data Engineering (AICDE - 2020), ISSN: 2666-285X. DOI: https://doi.org/10.1016/j.gltp.2021.01.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666285X21000042.

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

[3] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications*, pp. 64–74, 2008.

[4] J. Vanschoren, "Meta-learning," *Automated machine learning: methods, systems, challenges*, pp. 35–61, 2019.

[5] T. Bartz-Beielstein, C. Doerr, J. Bossek, *et al.*, "Benchmarking in optimization: Best practice and open issues," *CoRR*, vol. abs/2007.03488, 2020. arXiv: 2007.03488. [Online]. Available: https://arxiv.org/abs/2007.03488.

[6] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, *et al.*, "The FAIR guiding principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 1, pp. 1–9, 2016.

[7] 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, 2019.

[8] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown, "Performance prediction and automated tuning of randomized and parametric algorithms," in *Principles and Practice of Constraint Programming - CP 2006*, F. Benhamou, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 213–228, ISBN: 978-3-540-46268-2.

[9] J. M. Moyano, E. L. Gibaja, and S. Ventura, "MLDA: A tool for analyzing multi-label datasets," *Knowledge-Based Systems*, vol. 121, pp. 1–3, 2017.

[10] B. Smith, M. Ashburner, C. Rosse, *et al.*, "The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration," *Nature biotechnology*, vol. 25, no. 11, pp. 1251–1255, 2007.

[11] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Proc. of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4768–4777, ISBN: 9781510860964.

[12] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *International conference on machine learning*, PMLR, 2016, pp. 2071–2080.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[15] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNExplainer: Generating explanations for graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[16] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, and D. Nardi, *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[17] H. Bronkhorst, G. Roorda, C. Suhre, and M. Goedhart, "Logical reasoning in formal and everyday reasoning tasks," *International Journal of Science and Mathematics Education*, vol. 18, pp. 1673–1694, 2020.

[18] M. Morelli, M. Casagrande, and G. Forte, "Decision making: A theoretical review," *Integrative Psychological and Behavioral Science*, vol. 56, no. 3, pp. 609–629, 2022.

[19] T. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" *International journal of human-computer studies*, vol. 43, no. 5-6, pp. 907–928, 1995.

[20] O. Hartig, "Provenance information in the web of data," in *LDOW*, 2009.

[21] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "HermiT: an OWL 2 reasoner," *Journal of Automated Reasoning*, vol. 53, no. 3, pp. 245–269, 2014.

[22] D. Tsarkov and I. Horrocks, "FaCT++ description logic reasoner: System description," in *International joint conference on automated reasoning*, Seattle, USA: Springer, 2006, pp. 292–297.

[23] G. O. Consortium *et al.*, "Creating the gene ontology resource: Design and implementation," *Genome research*, vol. 11, no. 8, pp. 1425–1433, 2001.

[24] D. M. Dooley, E. J. Griffiths, G. S. Gosal, *et al.*, "Foodon: A harmonized food ontology to increase global food traceability, quality control and data integration," *npj Science of Food*, vol. 2, no. 1, pp. 1–10, 2018.

[25] P. L. Buttigieg, N. Morrison, B. Smith, C. J. Mungall, and S. E. Lewis, "The environment ontology: Contextualising biological and biomedical entities," *Journal of biomedical semantics*, vol. 4, no. 1, pp. 1–9, 2013.

[26] P. Panov, L. Soldatova, and S. Džeroski, "Ontology of core data mining entities," *Data Mining and Knowledge Discovery*, vol. 28, no. 5, pp. 1222–1265, 2014.

[27] F. van Harmelen, "The Semantic Web: What, Why, How, and When," *IEEE Distributed Systems Online*, vol. 5, no. 03, p. 4, Mar. 2004, ISSN: 1541-4922. DOI: 10.1109/MDSO.2004.1285880.

[28] O. Erling, "Virtuoso, a Hybrid RDBMS/Graph Column Store.," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 3–8, 2012.

[29] D. International, *DAMA-DMBOK: Data management body of knowledge*. Technics Publications, LLC, 2017.

[30] D. Lis, J. Gelhaar, and B. Otto, "Data strategy and policies: The role of data governance in data ecosystems," in *Data Governance: From the Fundamentals to Real Cases*, Springer, 2023, pp. 27–55.

[31] D. Lin, J. Crabtree, I. Dillo, *et al.*, "The TRUST Principles for digital repositories," *Scientific Data*, vol. 7, no. 1, pp. 1–5, 2020.

[32] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[33] D. Kocev, C. Vens, J. Struyf, and S. Džeroski, "Tree ensembles for predicting structured outputs," *Pattern Recognition*, vol. 46, no. 3, pp. 817–833, 2013.

[34] G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Džeroski, "An extensive experimental comparison of methods for multi-label learning," *Pattern recognition*, vol. 45, no. 9, pp. 3084–3104, 2012.

[35] J. Levatić, D. Kocev, and S. Džeroski, "The importance of the label hierarchy in hierarchical multi-label classification," *Journal of Intelligent Information Systems*, vol. 45, pp. 247–271, 2015.

[36] C. N. Silla and A. A. Freitas, "A survey of hierarchical classification across different application domains," *Data mining and knowledge discovery*, vol. 22, pp. 31–72, 2011.

[37] S. Dong, P. Wang, and K. Abbas, "A survey on deep learning and its applications," *Computer Science Review*, vol. 40, p. 100 379, 2021.

[38] S. B. Kotsiantis, "Decision trees: A recent overview," *Artificial Intelligence Review*, vol. 39, pp. 261–283, 2013.

[39] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[40] K. Gurney, *An introduction to neural networks*. CRC press, 2018.

[41] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining deep neural networks and beyond: A review of methods and applications," *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, 2021.

[42] R. Olson, W. La Cava, P. Orzechowski, R. Urbanowicz, and J. Moore, "PMLB: a large benchmark suite for machine learning evaluation and comparison," *BioData mining*, vol. 10, no. 1, p. 36, 2017.

[43] A. Rivolli, L. P. Garcia, C. Soares, J. Vanschoren, and A. C. de Carvalho, "Meta-features for meta-learning," *Knowledge-Based Systems*, vol. 240, p. 108 101, 2022.

[44] J. Bogatinovski, L. Todorovski, S. Džeroski, and D. Kocev, "Explaining the performance of multilabel classification methods with data set properties," *International Journal of Intelligent Systems*, vol. 37, no. 9, pp. 6080–6122, 2022.

[45] P. Kerschke, H. Hoos, F. Neumann, and H. Trautmann, "Automated Algorithm Selection: Survey and Perspectives," en, *Evolutionary Computation*, vol. 27, no. 1, pp. 3–45, Mar. 2019, ISSN: 1063-6560, 1530-9304. (visited on 12/11/2019).

[46] H. H. Hoos, "Automated algorithm configuration and parameter tuning," in *Autonomous search*, Springer, 2012, pp. 37–71.

[47] E. Chong, *An introduction to optimization*, 2013.

[48] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel, "Two decades of blackbox optimization applications," *EURO Journal on Computational Optimization*, vol. 9, p. 100 011, 2021.

[49] X.-S. Yang, "Metaheuristic optimization: Algorithm analysis and open problems," in *International symposium on experimental algorithms*, Springer, 2011, pp. 21–32.

[50] O. Kramer, *Genetic Algorithm Essentials* (Studies in Computational Intelligence). Springer, 2017, vol. 679, ISBN: 978-3-319-52155-8. DOI: 10.1007/978-3-319-52156-5. [Online]. Available: https://doi.org/10.1007/978-3-319-52156-5.

[51] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. of ICNN'95-international conference on neural networks*, IEEE, vol. 4, 1995, pp. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

[52] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[53] A. E. Eiben and M. Schoenauer, "Evolutionary computing," *Information Processing Letters*, vol. 82, no. 1, pp. 1–6, 2002.

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

[55] 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 *International Conference on Automated Machine Learning*, PMLR, 2023, pp. 7–1.

[56] J. Rapin and O. Teytaud, *Nevergrad - A gradient-free optimization platform*, https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[57] 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*, vol. 36, pp. 114–144, 2020. DOI: 10.1080/10556788.2020.1808977. [Online]. Available: https://doi.org/10.1080/10556788.2020.1808977.

[58] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph, "Exploratory Landscape Analysis," in *Genetic and Evolutionary Computation Conference (GECCO)*, ACM, 2011, pp. 829–836, ISBN: 978-1-4503-0557-0. (visited on 03/14/2019).

[59] 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.

[60] C. Song and R. Kawai, "Monte Carlo and variance reduction methods for structural reliability analysis: A comprehensive review," *Probabilistic Engineering Mechanics*, vol. 73, p. 103 479, 2023.

[61] 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: 978-3-030-25146-8.

[62] R. P. Prager and H. Trautmann, "Pflacco: Feature-based landscape analysis of continuous and constrained optimization problems in python," *Evolutionary Computation*, pp. 1–6, 2024.

[63] Q. Renau, C. Doerr, J. Dréo, and B. Doerr, "Exploratory landscape analysis is strongly sensitive to the sampling strategy," in *Proc. of Parallel Problem Solving from Nature (PPSN)*, ser. LNCS, Data available at: https://doi.org/10.5281/zenodo.3886816, vol. 12270, Springer, 2020, pp. 139–153. DOI: 10.1007/978-3-030-58115-2_10. [Online]. Available: https://doi.org/10.1007/978-3-030-58115-2%5C_10.

[64] T. Eftimov, A. Jankovic, G. Popovski, C. Doerr, and P. Korošec, "Personalizing performance regression models to black-box optimization problems," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO 2021)*, 2021.

[65] A. Jankovic, T. Eftimov, and C. Doerr, "Towards feature-based performance regression using trajectory data," in *Applications of Evolutionary Computation (EvoApplications 2021)*, Springer, vol. 12694, 2021, pp. 601–617.

[66] M. A. Muñoz, Y. Sun, M. Kirley, and S. K. Halgamuge, "Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges," *Inf. Sci.*, vol. 317, pp. 224–245, 2015. DOI: 10.1016/j.ins.2015.05.010.

[67] 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.

[68] G. Cenikj, A. Nikolikj, G. Petelin, N. van Stein, C. Doerr, and T. Eftimov, "A survey of meta-features used for automated selection of algorithms for black-box single-objective continuous optimization," *arXiv preprint arXiv:2406.06629*, 2024.

[69] G. Petelin, G. Cenikj, and T. Eftimov, "TLA: Topological landscape analysis for single-objective continuous optimization problem instances," in *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2022, pp. 1698–1705.

[70] L. Wasserman, "Topological data analysis," *Annual Review of Statistics and Its Application*, vol. 5, no. 1, pp. 501–532, 2018.

[71] R. P. Prager, M. V. 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)*, IEEE, 2021, pp. 1–8.

[72] 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 *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022, pp. 657–665.

[73] G. Cenikj, G. Petelin, and T. Eftimov, "TransOpt: Transformer-based representation learning for optimization problem classification," *arXiv preprint arXiv:2311.18035*, 2023.

[74] 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," *arXiv preprint arXiv:2401.01192*, 2024.

[75] 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*, 2023, pp. 515–518.

[76] 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.

[77] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proc. of IEEE Congress on Evolutionary Computation*, IEEE, 1996, pp. 312–317.

[78] 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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, ACM, 2021, pp. 1375–1384.

[79] D. Vermetten, F. Caraffini, A. V. Kononova, and T. Bäck, "Modular differential evolution," in *Proc. of the Genetic and Evolutionary Computation Conference*, ser. GECCO '23, ACM, 2023, pp. 864–872. DOI: 10.1145/3583131.3590417. [Online]. Available: https://doi.org/10.1145/3583131.3590417.

[80] S. Cahon, N. Melab, and E.-G. Talbi, "ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics," *Journal of Heuristics*, vol. 10, no. 3, pp. 357–380, 2004.

[81] J. Dreo, A. Liefooghe, S. Verel, *et al.*, "ParadisEO: From a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of paradiseo," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, ACM, 2021, pp. 1522–1530.

[82] C. L. Camacho-Villalón, M. Dorigo, and T. Stützle, "PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 3, pp. 402–416, 2021.

[83] R. Boks, H. Wang, and T. Bäck, "A modular hybridization of particle swarm optimization and differential evolution," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, ACM, 2020, pp. 1418–1425.

[84] J. R. Rice, "The algorithm selection problem," in *Advances in computers*, vol. 15, Elsevier, 1976, pp. 65–118.

[85] M. A. Muñoz, Y. Sun, M. Kirley, and S. K. Halgamuge, "Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges," *Information Sciences*, vol. 317, pp. 224–245, 2015.

[86] L. Meunier, H. Rakotoarison, P. K. Wong, *et al.*, "Black-box optimization revisited: Improving algorithm selection wizards through massive benchmarking," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 3, pp. 490–500, 2021.

[87] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowledge-based systems*, vol. 212, p. 106 622, 2021.

[88] U. Škvorc, T. Eftimov, and P. Korošec, "Transfer learning analysis of multi-class classification for landscape-aware algorithm selection," *Mathematics*, vol. 10, no. 3, p. 432, 2022.

[89] 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.

[90] Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, "Algorithm portfolios based on cost-sensitive hierarchical clustering.," in *IJCAI*, vol. 13, 2013, pp. 608–614.

[91] J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren, "Fast algorithm selection using learning curves," in *Advances in Intelligent Data Analysis XIV: 14th International Symposium, IDA 2015, Saint Etienne. France, October 22-24, 2015. Proceedings 14*, Springer, 2015, pp. 298–309.

[92] D. Pulatov, M. Anastacio, L. Kotthoff, and H. Hoos, "Opening the black box: Automated software analysis for algorithm selection," in *International Conference on Automated Machine Learning*, PMLR, 2022, pp. 6–1.

[93] M. Alissa, K. Sim, and E. Hart, "Automated algorithm selection: From feature-based to feature-free approaches," *Journal of Heuristics*, vol. 29, no. 1, pp. 1–38, 2023.

[94] L. Tian, X. Zhou, Y.-P. Wu, W.-T. Zhou, J.-H. Zhang, and T.-S. Zhang, "Knowledge graph and knowledge reasoning: A systematic review," *Journal of Electronic Science and Technology*, vol. 20, no. 2, p. 100 159, 2022.

[95] T. Shen, F. Zhang, and J. Cheng, "A comprehensive overview of knowledge graph completion," *Knowledge-Based Systems*, vol. 255, p. 109 597, 2022.

[96] D. Lee, B. Oh, S. Seo, and K.-H. Lee, "News recommendation with topic-enriched knowledge graphs," in *Proceedings of the 29th ACM international conference on information & knowledge management*, 2020, pp. 695–704.

[97] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, "Product knowledge graph embedding for e-commerce," in *Proceedings of the 13th international conference on web search and data mining*, 2020, pp. 672–680.

[98] F. Gong, M. Wang, H. Wang, S. Wang, and M. Liu, "SMR: medical knowledge graph embedding for safe medicine recommendation," *Big Data Research*, vol. 23, p. 100 174, 2021.

[99] Z. Gun and J. Chen, "Novel knowledge graph-and knowledge reasoning-based classification prototype for obia using high resolution remote sensing imagery," *Remote Sensing*, vol. 15, no. 2, p. 321, 2023.

[100] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," in *International Conference on Learning Representations*, 2014. [Online]. Available: `https://api.semanticscholar.org/CorpusID:2768038`.

[101] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[102] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[103] A. Kostovska, J. Bogatinovski, S. Džeroski, D. Kocev, and P. Panov, "A catalogue with semantic annotations makes multilabel datasets FAIR," *Scientific Reports*, vol. 12, no. 1, p. 7267, 2022.

[104] F. Charte, A. J. Rivera, D. Charte, M. J. del Jesus, and F. Herrera, "Tips, guidelines and tools for managing multi-label datasets: The MLDR datasets R package and the Cometa data repository," *Neurocomputing*, vol. 289, pp. 68–85, 2018.

[105] *Cometa*, Available at: https://cometa.ujaen.es/datasets/, 2024. [Online]. Available: https://cometa.ujaen.es/datasets/.

[106] *KDIS-Cordoba*, Available at: https://www.uco.es/kdis/mllresources//, 2024. [Online]. Available: https://www.uco.es/kdis/mllresources/.

[107] *MULAN*, Available at: https://mulan.sourceforge.net/datasets-mlc.html, 2024. [Online]. Available: https://mulan.sourceforge.net/datasets-mlc.html.

[108] K. Bhatia, K. Dahiya, H. Jain, *et al.*, *The extreme classification repository: Multi-label datasets and code*, Available at: http://manikvarma.org/downloads/XC/XMLRepository.html, 2016. [Online]. Available: http://manikvarma.org/downloads/XC/XMLRepository.html.

[109] N. Macia and E. Bernadó-Mansilla, "Towards UCI+: a mindful repository design," *Information Sciences*, vol. 261, pp. 237–262, 2014.

[110] J. Rijn, B. Bischl, L. Torgo, *et al.*, "OpenML: A collaborative science platform," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Prague, Czech Republic: Springer, 2013, pp. 645–649.

[111] M.-L. Alvite-Diez, "Linked open data portals: Functionalities and user experience in semantic catalogues," *Online Information Review*, vol. 45, no. 5, pp. 946–963, 2021.

[112] S. Kim and E. E. Bolton, "PubChem: A Large-Scale Public Chemical Database for Drug Discovery," *Open Access Databases and Datasets for Drug Discovery*, pp. 39–66, 2024.

[113] H. Dibowski, S. Schmid, Y. Svetashova, C. Henson, and T. Tran, "Using semantic technologies to manage a data lake: Data catalog, provenance and access control.," in *SSWS@ ISWC*, Athen, 2020, pp. 65–80.

[114] N. Kasrin, M. Qureshi, S. Steuer, and D. Nicklas, "Semantic data management for experimental manufacturing technologies," *Datenbank-Spektrum*, vol. 18, pp. 27–37, 2018.

[115] A. Kostovska, S. Džeroski, and P. Panov, "Semantic description of data mining datasets: An ontology-based annotation schema," in *Proceedings of International Conference on Discovery Science*, Springer, 2020, pp. 140–155.

[116] *List of Schema.org Dataset properties*, Available at: https://schema.org/Dataset, 2021. [Online]. Available: https://schema.org/Dataset.

[117] *Schema.org*, Available at: https://schema.org/, 2024. [Online]. Available: https://schema.org/.

[118] P. Panov, L. N. Soldatova, and S. Džeroski, "Generic ontology of datatypes," *Information Sciences*, vol. 329, pp. 900–920, 2016.

[119] M. Keet, A. Lawrynowicz, C. d'Amato, *et al.*, "The data mining OPtimization ontology," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 32, pp. 43–53, 2015.

[120] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "Mulan: A java library for multi-label learning," *The Journal of Machine Learning Research*, vol. 12, pp. 2411–2414, 2011.

[121] F. Briggs, B. Lakshminarayanan, L. Neal, *et al.*, "Acoustic classification of multiple simultaneous bird species: A multi-instance multi-label approach," *The Journal of the Acoustical Society of America*, vol. 131, no. 6, pp. 4640–4650, 2012.

[122] I. Tolovski, S. Džeroski, and P. Panov, "Semantic annotation of predictive modelling experiments," in *Discovery Science: 23rd International Conference, DS 2020, Thessaloniki, Greece, October 19–21, 2020, Proceedings 23*, Springer, 2020, pp. 124–139.

[123] A. Bandrowski, R. Brinkman, M. Brochhausen, *et al.*, "The ontology for biomedical investigations," *PloS one*, vol. 11, no. 4, e0154556, 2016.

[124] J. Bogatinovski, L. Todorovski, S. Džeroski, and D. Kocev, "Comprehensive comparative study of multi-label classification methods," *Expert Systems with Applications*, vol. 203, p. 117 215, 2022.

[125] *Weka ARFF file format*, Available at: `https://waikato.github.io/weka-wiki/arff/`, 2021. [Online]. Available: `https://waikato.github.io/weka-wiki/arff/`.

[126] *Apache Jena RDF API*, Available at: `https://jena.apache.org/documentation/rdf/index.html`, 2024. [Online]. Available: `https://jena.apache.org/documentation/rdf/index.html`.

[127] *Apache Jena Library*, Available at: `https://jena.apache.org/index.html`, 2024. [Online]. Available: `https://jena.apache.org/index.html`.

[128] *Apache Jena Fuseki server*, Available at: `https://jena.apache.org/documentation/fuseki2/`, 2021. [Online]. Available: `https://jena.apache.org/documentation/fuseki2/`.

[129] *Apache Jena Inference Support*, Available at: `https://jena.apache.org/documentation/inference/index.html`, 2024. [Online]. Available: `https://jena.apache.org/documentation/inference/index.html`.

[130] A. Kostovska, D. Vermetten, C. Doerr, S. Džeroski, P. Panov, and T. Eftimov, "OPTION: OPTImization Algorithm Benchmarking ONtology," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 6, pp. 1618–1632, 2023. DOI: `10.1109/TEVC.2022.3232844`.

[131] A. Kostovska, D. Vermetten, C. Doerr, S. Džeroski, P. Panov, and T. Eftimov, "OPTION: OPTImization Algorithm Benchmarking ONtology," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2021, pp. 239–240.

[132] F. Caraffini and G. Iacca, "The SOS Platform: Designing, Tuning and Statistically Benchmarking Optimisation Algorithms," *Mathematics*, vol. 8, no. 5, 2020, ISSN: 2227-7390. DOI: `10.3390/math8050785`. [Online]. Available: `https://www.mdpi.com/2227-7390/8/5/785`.

[133] O. E. Gundersen and S. Kjensmo, "State of the art: Reproducibility in artificial intelligence," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[134] R. E. Carter, Z. I. Attia, F. Lopez-Jimenez, and P. A. Friedman, "Pragmatic considerations for fostering reproducible research in artificial intelligence," *NPJ digital medicine*, vol. 2, no. 1, pp. 1–3, 2019.

[135] M. López-Ibáñez, J. Branke, and L. Paquete, "Reproducibility in evolutionary computation," *ACM Trans. Evol. Learn. Optim.*, vol. 1, no. 4, 14:1–14:21, 2021. DOI: `10.1145/3466624`. [Online]. Available: `https://doi.org/10.1145/3466624`.

[136] C. Doerr, H. Wang, F. Ye, S. van Rijn, and T. Bäck, "IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics," *CoRR*, vol. abs/1810.05281, 2018, An up-to-date documentation of IOHprofiler is available at `https://iohprofiler.github.io/`. [Online]. Available: `http://arxiv.org/abs/1810.05281`.

[137] C. L. Camacho-Villalón, T. Stützle, and M. Dorigo, "Grey wolf, firefly and bat algorithms: Three widespread algorithms that do not contain any novelty," in *Proc. of Swarm Intelligence (ANTS'20)*, ser. LNCS, vol. 12421, Springer, 2020, pp. 121–133. DOI: 10.1007/978-3-030-60376-2_10. [Online]. Available: https://doi.org/10.1007/978-3-030-60376-2%5C_10.

[138] K. Sörensen, "Metaheuristics - the metaphor exposed," *International Transactions in Operational Research (ITOR)*, vol. 22, pp. 3–18, 2015.

[139] A. Yaman, A. Hallawa, M. Coler, and G. Iacca, "Presenting the ECO: evolutionary computation ontology," in *European conference on the applications of evolutionary computation*, Springer, 2017, pp. 603–619.

[140] V. Basto-Fernandes, I. Yevseyeva, A. Deutz, and M. Emmerich, "A survey of diversity oriented optimization: Problems, indicators, and algorithms," in *EVOLVE–A Bridge between Probability, Set Oriented Numerics and Evolutionary Computation VII*, Springer, 2017, pp. 3–23.

[141] L. Li, I. Yevseyeva, V. Basto-Fernandes, H. Trautmann, N. Jing, and M. Emmerich, "Building and using an ontology of preference-based multiobjective evolutionary algorithms," in *International conference on evolutionary multi-criterion optimization*, Springer, 2017, pp. 406–421.

[142] B. Smith *et al.*, "Relations in biomedical ontologies," *Genome biology*, vol. 6, no. 5, R46, 2005.

[143] R. Arp, B. Smith, and A. Spear, *Building ontologies with basic formal ontology*. Mit Press, 2015.

[144] M. Dumontier, C. J. Baker, J. Baran, *et al.*, "The semanticscience integrated ontology (sio) for biomedical research and knowledge discovery," *Journal of biomedical semantics*, vol. 5, no. 1, pp. 1–11, 2014.

[145] N. F. Noy, M. Crubézy, R. W. Fergerson, *et al.*, "Protégé-2000: An open-source ontology-development and knowledge-acquisition environment.," in *AMIA... Annual Symposium proceedings. AMIA Symposium*, American Medical Informatics Association, vol. 2003, 2003, pp. 953–953.

[146] N. F. Noy, N. H. Shah, P. L. Whetzel, *et al.*, "BioPortal: ontologies and integrated data resources at the click of a mouse," *Nucleic acids research*, vol. 37, no. suppl_2, W170–W173, 2009.

[147] A. Lawrynowicz, D. Esteves, P. Panov, T. Soru, S. Dzeroski, and J. Vanschoren, "An algorithm, implementation and execution ontology design pattern," *Adv. Ontol. Des. Patterns*, vol. 32, p. 55, 2017.

[148] M. A. Musen, "The protégé project: A look back and a look forward," *AI matters*, vol. 1, no. 4, pp. 4–12, 2015.

[149] D. C. M. Initiative *et al.*, *Dublin core metadata element set, version 1.1*, 2012.

[150] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Posík, "Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009," in *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010, Companion Material*, M. Pelikan and J. Branke, Eds., ACM, 2010, pp. 1689–1696. DOI: 10.1145/1830761.1830790. [Online]. Available: https://doi.org/10.1145/1830761.1830790.

[151] N. Hansen, A. Auger, and D. Brockhoff, *Data repository of the BBOB test suite of the COCO benchmark environment*, https://numbbo.github.io/data-archive/bbob/, Data collected from 2010 to 2020, 2020.

[152] J. Liu, A. Moreau, M. Preuss, *et al.*, "Versatile black-box optimization," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 620–628.

[153] Q. Renau, C. Doerr, J. Dréo, and B. Doerr, "Exploratory landscape analysis is strongly sensitive to the sampling strategy," in *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, ser. LNCS, vol. 12270, Springer, 2020, pp. 139–153. DOI: 10.1007/978-3-030-58115-2_10. [Online]. Available: https://doi.org/10.1007/978-3-030-58115-2%5C_10.

[154] J. de Nobel, F. Ye, D. Vermetten, H. Wang, C. Doerr, and T. Bäck, "IOHexperimenter: Benchmarking platform for iterative optimization heuristics," *Evolutionary Computation*, pp. 1–6, 2024, Available online at https://doi.org/10.1162/evco_a_00342. DOI: 10.1162/evco_a_00342. eprint: https://direct.mit.edu/evco/article-pdf/doi/10.1162/evco_a_00342/2335957/evco_a_00342.pdf. [Online]. Available: https://doi.org/10.1162/evco%5C_a%5C_00342.

[155] *SPARQL*, Available at: https://www.w3.org/TR/rdf-sparql-query/, 2021. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/.

[156] L. Richardson and S. Ruby, *RESTful web services*. " O'Reilly Media, Inc.", 2008.

[157] H. Wang, D. Vermetten, F. Ye, C. Doerr, and T. Bäck, "IOHanalyzer: Detailed performance analyses for iterative optimization heuristics," *ACM Transactions on Evolutionary Learning and Optimization*, 2022, ISSN: 2688-299X. DOI: 10.1145/3510426. [Online]. Available: https://doi.org/10.1145/3510426.

[158] H. Stegherr, M. Heider, and J. Hähner, "Classifying metaheuristics: Towards a unified multi-level classification system," *Natural Computing*, pp. 1–17, 2020.

[159] J. Stork, A. E. Eiben, and T. Bartz-Beielstein, "A new taxonomy of global optimization algorithms," *Natural Computing: An International Journal*, vol. 21, no. 2, pp. 219–242, Jun. 2022, ISSN: 1567-7818. DOI: 10.1007/s11047-020-09820-4. [Online]. Available: https://doi.org/10.1007/s11047-020-09820-4.

[160] J. Liu, S. Anavatti, M. Garratt, K. C. Tan, and H. A. Abbass, "A survey, taxonomy and progress evaluation of three decades of swarm optimisation," *Artificial Intelligence Review*, pp. 1–119, 2021.

[161] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[162] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *J. Artif. Intell. Res.*, vol. 32, pp. 565–606, 2008. DOI: 10.1613/jair.2490. [Online]. Available: https://doi.org/10.1613/jair.2490.

[163] Z. Mu, H. H. Hoos, and T. Stützle, "The impact of automated algorithm configuration on the scaling behaviour of state-of-the-art inexact tsp solvers," in *Learning and Intelligent Optimization*, P. Festa, M. Sellmann, and J. Vanschoren, Eds., Springer International Publishing, 2016, pp. 157–172, ISBN: 978-3-319-50349-3. DOI: 10.1007/978-3-319-50349-3_11. [Online]. Available: https://doi.org/10.1007/978-3-319-50349-3_11.

[164] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. H. Hoos, and K. Leyton-Brown, "The configurable SAT solver challenge (CSSC)," *Artif. Intell.*, vol. 243, pp. 1–25, 2017. DOI: 10.1016/j.artint.2016.09.006. [Online]. Available: https://doi.org/10.1016/j.artint.2016.09.006.

[165] P. Kerschke, L. Kotthoff, J. Bossek, H. H. Hoos, and H. Trautmann, "Leveraging TSP solver complementarity through machine learning," *Evol. Comput.*, vol. 26, no. 4, 2018. DOI: 10.1162/evco_a_00215. [Online]. Available: https://doi.org/10.1162/evco%5C_a%5C_00215.

[166] C. Molnar, *Interpretable Machine Learning*. Lulu Press, 2020.

[167] H. Chen, S. M. Lundberg, and S.-I. Lee, "Explaining a series of models by propagating Shapley values," *Nature communications*, vol. 13, no. 1, p. 4512, 2022.

[168] I. E. Kumar, S. Venkatasubramanian, C. Scheidegger, and S. Friedler, "Problems with Shapley-value-based explanations as feature importance measures," in *International Conference on Machine Learning*, PMLR, 2020, pp. 5491–5500.

[169] 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*, 2023, pp. 495–498.

[170] A. Tornede, L. Gehring, T. Tornede, M. Wever, and E. Hüllermeier, "Algorithm selection on a meta level," *Machine Learning*, pp. 1–34, 2022.

[171] S. Ali and K. A. Smith, "On learning algorithm selection for classification," *Applied Soft Computing*, vol. 6, no. 2, pp. 119–138, 2006.

[172] N. Pise and P. Kulkarni, "Algorithm selection for classification problems," in *2016 SAI Computing Conference (SAI)*, 2016, pp. 203–211. DOI: 10.1109/SAI.2016.7555983.

[173] N. Cohen-Shapira and L. Rokach, "Automatic selection of clustering algorithms using supervised graph embedding," *Information Sciences*, vol. 577, pp. 824–851, 2021.

[174] L. Xu, F. Hutter, J. Shen, H. H. Hoos, and K. Leyton-Brown, "SATzilla2012: Improved algorithm selection based on cost-sensitive classification models," *Proceedings of SAT Challenge*, pp. 57–58, 2012.

[175] A. Kostovska, G. Cenikj, D. Vermetten, *et al.*, "PS-AAS: Portfolio Selection for Automated Algorithm Selection in Black-Box Optimization," in *International Conference on Automated Machine Learning*, PMLR, 2023, pp. 11–1.

[176] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[177] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine Learning*, vol. 85, p. 333, 2011.

[178] R. Schapire and Y. Singer, "Boostexter: A boosting-based system for text categorization," *Machine Learning*, vol. 39, pp. 135–168, 2000.

[179] G. Nasierding, A. Kouzani, and G. Tsoumakas, "A triple-random ensemble classification method for mining multi-label data," in *IEEE International Conference on Data Mining Workshops*, Washington, DC, USA: IEEE Computer Society, 2010, pp. 49–56.

[180] K. Brinker, "On active learning in multi-label classification," in *From Data and Information Analysis to Knowledge Engineering*, Berlin, Heidelberg: Springer, 2006, pp. 206–213.

[181] J. Fürnkranz, E. Hüllermeier, E. Loza Mencía, and K. Brinker, "Multilabel classification via calibrated label ranking," *Machine Learning*, vol. 73, no. 2, pp. 133–153, 2008.

[182] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining*, vol. 2007, pp. 1–13, 2007.

[183] J. Read, B. Pfahringer, and G. Holmes, "Multi-label Classification Using Ensembles of Pruned Sets," in *Proceedings of the 8th IEEE International Conference on Data Mining*, Washington, DC, USA: IEEE Computer Society, 2008, pp. 995–1000.

[184] J. Read, "Scalable multi-label classification," Ph.D. dissertation, University of Waikato, Hamilton, New Zeland, 2010.

[185] 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.

[186] A. Kostovska, D. Vermetten, P. Korošec, S. Džeroski, C. Doerr, and T. Eftimov, "Using machine learning methods to assess module performance contribution in modular optimization frameworks," *Evolutionary Computation*, pp. 1–27, Aug. 2024, ISSN: 1063-6560. DOI: 10.1162/evco_a_00356.

[187] 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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ACM, 2022, pp. 648–656.

[188] K. Hussain, M. N. Mohd Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: A comprehensive survey," *Artificial intelligence review*, vol. 52, pp. 2191–2233, 2019.

[189] C. Aranha, C. L. Camacho Villalón, F. Campelo, *et al.*, "Metaphor-based metaheuristics, a call for action: The elephant in the room," *Swarm Intelligence*, vol. 16, no. 1, pp. 1–6, 2022.

[190] T. Eftimov, P. Korošec, and B. K. Seljak, "A novel approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics," *Information Sciences*, vol. 417, pp. 186–215, 2017.

[191] J. N. Hooker, "Needed: An empirical science of algorithms," *Operations research*, vol. 42, no. 2, pp. 201–212, 1994.

[192] J. N. Hooker, "Testing heuristics: We have it all wrong," *Journal of Heuristics*, vol. 1, pp. 33–42, 1995.

[193] N. G. Hall and M. E. Posner, "The generation of experimental data for computational testing in optimization," in *Experimental methods for the analysis of optimization algorithms*, Springer, 2010, pp. 73–101.

[194] M. A. Lones, "Mitigating metaphors: A comprehensible guide to recent nature-inspired algorithms," *SN Computer Science*, vol. 1, no. 1, p. 49, 2020.

[195] H. Stegherr, M. Heider, and J. Hähner, "Classifying Metaheuristics: Towards a unified multi-level classification system," *Natural Computing*, vol. 21, no. 2, pp. 155–171, 2022.

[196] B. Andersen, G. Delipei, D. Kropaczek, and J. Hou, "MOF: A modular framework for rapid application of optimization methodologies to general engineering design problems," *arXiv preprint arXiv:2204.00141*, 2022.

[197] A. Nikolikj, A. Kostovska, D. Vermetten, C. Doerr, and T. Eftimov, "Quantifying individual and joint module impact in modular optimization frameworks," *arXiv preprint arXiv:2405.11964*, 2024.

[198] D. Vermetten, M. López-Ibáñez, O. Mersmann, R. Allmendinger, and A. V. Kononova, "Analysis of modular CMA-ES on strict box-constrained problems in the SBOX-COST benchmarking suite," in *Proc. of the Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, ACM, 2023, pp. 2346–2353. DOI: 10.1145/3583133.3596419. [Online]. Available: https://doi.org/10.1145/3583133.3596419.

[199] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE transactions on evolutionary computation*, vol. 15, no. 1, pp. 4–31, 2010.

[200] R. Trajanov, S. Dimeski, M. Popovski, P. Korošec, and T. Eftimov, "Explainable landscape-aware optimization performance prediction," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2021, pp. 01–08.

[201] A. Nikolikj, R. Lang, P. Korošec, and T. Eftimov, "Explaining differential evolution performance through problem landscape characteristics," in *Proc. of Bioinspired Optimization Methods and Their Applications (BIOMA))*, Springer, 2022, pp. 99–113.

[202] A. Jankovic and C. Doerr, "Landscape-aware fixed-budget performance regression and algorithm selection for modular CMA-ES variants," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ACM, 2020, pp. 841–849. DOI: 10.1145/3377930.3390183. [Online]. Available: https://doi.org/10.1145/3377930.3390183.

[203] R. P. Prager, H. Trautmann, H. Wang, T. H. Bäck, and P. Kerschke, "Per-Instance Configuration of the Modularized CMA-ES by Means of Classifier Chains and Exploratory Landscape Analysis," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2020, pp. 996–1003.

[204] T. Eftimov, G. Popovski, D. Kocev, and P. Korošec, "Performance2vec: A step further in explainable stochastic optimization algorithm performance," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, ACM, 2020, pp. 193–194.

[205] 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.

[206] B. Rozemberczki, L. Watson, P. Bayer, *et al.*, "The Shapley value in machine learning," in *The 31st International Joint Conference on Artificial Intelligence and the 25th European Conference on Artificial Intelligence*, 2022.

[207] J. de Nobel, H. Wang, and T. Baeck, "Explorative data analysis of time series based algorithm features of CMA-ES variants," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, 2021, pp. 510–518.

[208] L. C. Bezerra, M. López-Ibánez, and T. Stützle, "Automatic component-wise design of multiobjective evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 403–417, 2015.

[209] A. Aziz-Alaoui, C. Doerr, and J. Dreo, "Towards large scale automated algorithm design by integrating modular benchmarking frameworks," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, ACM, 2021, pp. 1365–1374.

[210] T. Weise and Z. Wu, "Difficult features of combinatorial optimization problems and the tunable W-model benchmark problem for simulating them," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, ACM, 2018, pp. 1769–1776.

[211] C. Doerr, F. Ye, N. Horesh, H. Wang, O. M. Shir, and T. Bäck, "Benchmarking discrete optimization heuristics with IOHprofiler," *Applied Soft Computing*, vol. 88, p. 106 027, 2020. DOI: 10.1016/j.asoc.2019.106027. [Online]. Available: https://doi.org/10.1016/j.asoc.2019.106027.

[212] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[213] D. Rey and M. Neuhäuser, "Wilcoxon-signed-rank test," in *International Encyclopedia of Statistical Science*, M. Lovric, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1658–1659, ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_616. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_616.

[214] P. Kerschke and H. Trautmann, "The R-package FLACCO for exploratory landscape analysis with applications to multi-objective optimization problems," in *Proc. of IEEE Congress on Evolutionary Computation*, IEEE, 2016, pp. 5262–5269. DOI: 10.1109/CEC.2016.7748359. [Online]. Available: https://doi.org/10.1109/CEC.2016.7748359.

[215] Q. Renau, J. Dréo, C. Doerr, and B. Doerr, "Towards Explainable Exploratory Landscape Analysis: Extreme Feature Selection for Classifying BBOB Functions," 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_2. [Online]. Available: https://doi.org/10.1007/978-3-030-72699-7%5C_2.

[216] 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 *Proc. of IEEE Congress on Evolutionary Computation*, IEEE, 2022, pp. 1–8.

[217] M. A. Muñoz, M. Kirley, and S. K. Halgamuge, "A meta-learning prediction model of algorithm performance for continuous optimization problems," in *Proc. of Parallel Problem Solving from Nature (PPSN)*, Springer, 2012, pp. 226–235.

[218] M. Collautti, Y. Malitsky, D. Mehta, and B. O'Sullivan, "SNNAP: Solver-Based Nearest Neighbor for Algorithm Portfolios," in *Machine Learning and Knowledge Discovery in Databases*, Springer, 2013, pp. 435–450, ISBN: 978-3-642-40994-3.

[219] P. Kerschke and H. Trautmann, "Automated Algorithm Selection on Continuous Black-Box Problems by Combining Exploratory Landscape Analysis and Machine Learning," *Evolutionary Computation*, vol. 27, no. 1, pp. 99–127, 2019. DOI: 10.1162/evco_a_00236. [Online]. Available: https://doi.org/10.1162/evco%5C_a%5C_00236.

[220] A. Kostovska, A. Jankovic, D. Vermetten, *et al.*, "Per-run algorithm selection with warm-starting using trajectory-based features," in *Proc. of Parallel Problem Solving from Nature (PPSN)*, Springer, 2022, pp. 46–60.

[221] A. Jankovic, G. Popovski, T. Eftimov, and C. Doerr, "The Impact of Hyper-Parameter Tuning for Landscape-Aware Performance Regression and Algorithm Selection," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ACM, 2021.

[222] S. Varma and R. Simon, "Bias in error estimation when using cross-validation for model selection," *BMC bioinformatics*, vol. 7, no. 1, pp. 1–8, 2006.

[223] S. Bates, T. Hastie, and R. Tibshirani, "Cross-validation: What does it estimate and how well does it do it?" *Journal of the American Statistical Association*, pp. 1–12, 2023.

[224] 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*, 2023. [Online]. Available: https://openreview.net/forum?id=cp5PvcI6w8_.

[225] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Posík, "Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material)*, ACM, 2010, pp. 1689–1696. DOI: 10.1145/1830761.1830790. [Online]. Available: https://doi.org/10.1145/1830761.1830790.

[226] A. P. Piotrowski, "Review of differential evolution population size," *Swarm Evol. Comput.*, vol. 32, pp. 1–24, 2017. DOI: 10.1016/j.swevo.2016.05.003. [Online]. Available: https://doi.org/10.1016/j.swevo.2016.05.003.

[227] A. Kostovska, D. Vermetten, P. Korošec, S. Džeroski, C. Doerr, and T. Eftimov, *Linking Problem Features with Configurations of Modular Black-box Optimization Algorithms*, Data, code, additional figures for this work are available at https://doi.org/10.5281/zenodo.8151814, Jun. 2023. DOI: 10.5281/zenodo.8151814. [Online]. Available: https://doi.org/10.5281/zenodo.8151814.

[228] F. W. Scholz and M. A. Stephens, "K-sample anderson–darling tests," *Journal of the American Statistical Association*, vol. 82, no. 399, pp. 918–924, 1987.

[229] L. McInnes, J. Healy, N. Saul, and L. Großberger, "UMAP: Uniform Manifold Approximation and Projection," *Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018. DOI: 10.21105/joss.00861. [Online]. Available: https://doi.org/10.21105/joss.00861.

[230] 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.

[231] X. Chen, S. Jia, and Y. Xiang, "A review: Knowledge reasoning over knowledge graph," *Expert Systems with Applications*, vol. 141, p. 112 948, 2020.

[232] P. Wang, H. Jiang, J. Xu, and Q. Zhang, "Knowledge graph construction and applications for web search and beyond," *Data Intelligence*, vol. 1, no. 4, pp. 333–349, 2019.

[233] X. Zhao, H. Chen, Z. Xing, and C. Miao, "Brain-inspired search engine assistant based on knowledge graph," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 8, pp. 4386–4400, 2021.

[234] X. Zeng, X. Tu, Y. Liu, X. Fu, and Y. Su, "Toward better drug discovery with knowledge graph," *Current opinion in structural biology*, vol. 72, pp. 114–126, 2022.

[235] F. MacLean, "Knowledge graphs and their applications in drug discovery," *Expert opinion on drug discovery*, vol. 16, no. 9, pp. 1057–1069, 2021.

[236] J. Qian, X.-Y. Li, C. Zhang, L. Chen, T. Jung, and J. Han, "Social network deanonymization and privacy inference with knowledge graph model," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 679–692, 2017.

[237] Z. Wang, T. Chen, J. S. J. Ren, W. Yu, H. Cheng, and L. Lin, "Deep reasoning with knowledge graph for social relationship understanding," in *International Joint Conference on Artificial Intelligence*, 2018. [Online]. Available: `https://api.semanticscholar.org/CorpusID:49558620`.

[238] D. M. Bean, H. Wu, E. Iqbal, *et al.*, "Knowledge graph prediction of unknown adverse drug reactions and validation in electronic health records," *Scientific reports*, vol. 7, no. 1, p. 16 416, 2017.

[239] X. Tao, T. Pham, J. Zhang, *et al.*, "Mining health knowledge graph for health risk prediction," *World Wide Web*, vol. 23, pp. 2341–2362, 2020.

[240] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011.

[241] A. H. Halim, I. Ismail, and S. Das, "Performance assessment of the metaheuristic optimization algorithms: An exhaustive review," *Artificial Intelligence Review*, vol. 54, no. 3, pp. 2323–2409, 2021.

[242] G. Mahmoudi and C. Muller-Schloer, "Semantic multi-criteria decision making semcdm," in *2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM)*, IEEE, 2009, pp. 149–156.

[243] J. F. Aldana-Martín, M. del Mar Roldán-García, A. J. Nebro, and J. F. Aldana-Montes, "MOODY: An ontology-driven framework for standardizing multi-objective evolutionary algorithms," *Information Sciences*, vol. 661, p. 120 184, 2024.

[244] X. Xue and H. Zhu, "Matching knowledge graphs with compact niching evolutionary algorithm," *Expert Systems with Applications*, vol. 203, p. 117 371, 2022.

[245] X. Xue, "Automatic knowledge graph matching via self-adaptive designed genetic programming," *Knowledge-Based Systems*, vol. 293, p. 111 628, 2024.

[246] Z. Liu, D. Yang, Y. Wang, M. Lu, and R. Li, "EGNN: Graph structure learning based on evolutionary computation helps more in graph neural networks," *Applied Soft Computing*, vol. 135, p. 110 040, 2023.

[247] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[248] L. Costabello, S. Pai, C. Le Van, R. McGrath, N. McCarthy, and P. Tabacof, "Ampligraph: A library for representation learning on knowledge graphs," *Retrieved Oct*, vol. 10, p. 2019, 2019.

[249] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. AISTATS*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[250] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[251] R. Celebi, H. Uyar, E. Yasar, O. Gumus, O. Dikenelli, and M. Dumontier, "Evaluation of knowledge graph embedding approaches for drug-drug interaction prediction in realistic settings," *BMC bioinformatics*, vol. 20, pp. 1–14, 2019.

[252] G. Lachaud, P. Conde-Cespedes, and M. Trocan, "Comparison between inductive and transductive learning in a real citation network using graph neural networks," in *2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2022, pp. 534–540. DOI: 10.1109/ASONAM55673.2022.10068589.

[253] J. Lukasik, D. Friede, H. Stuckenschmidt, and M. Keuper, "Neural architecture performance prediction using graph neural networks," in *Pattern Recognition: 42nd DAGM German Conference, DAGM GCPR 2020, Tübingen, Germany, September 28–October 1, 2020, Proceedings 42*, Springer, 2021, pp. 188–201.

[254] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *International conference on machine learning*, PMLR, 2019, pp. 7105–7114.

[255] S. Singh, B. Steiner, J. Hegarty, and H. Leather, "Using graph neural networks to model the performance of deep neural networks," *arXiv preprint arXiv:2108.12489*, 2021.

[256] Y. Chai, D. Tripathy, C. Zhou, *et al.*, "Perfsage: Generalized inference performance predictor for arbitrary deep learning models on edge devices," *arXiv preprint arXiv:2301.10999*, 2023.

[257] Y. Sun and J. Han, "Mining heterogeneous information networks: A structural analysis approach," *SIGKDD Explor. Newsl.*, vol. 14, no. 2, pp. 20–28, Apr. 2013, ISSN: 1931-0145. DOI: 10.1145/2481244.2481248. [Online]. Available: https://doi.org/10.1145/2481244.2481248.

[258] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in neural information processing systems*, vol. 32, 2019.

[259] M. Y. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *ICLR workshop on representation learning on graphs and manifolds*, 2019.

[260] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[261] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," *arXiv preprint arXiv:1606.08415*, 2016.

[262] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 2171–2175, 2012.

[263] W. Fischl, G. Gottlob, D. M. Longo, and R. Pichler, "Hyperbench: A benchmark and tool for hypergraphs and empirical findings," *Journal of Experimental Algorithmics (JEA)*, vol. 26, pp. 1–40, 2021.

[264] T. Eftimov, G. Petelin, G. Cenikj, *et al.*, "Less is more: Selecting the right benchmarking set of data for time series classification," *Expert Systems with Applications*, vol. 198, p. 116 871, 2022.

[265] G. Cenikj, R. D. Lang, A. P. Engelbrecht, C. Doerr, P. Korošec, and T. Eftimov, "SELECTOR: selecting a representative benchmark suite for reproducible statistical comparison," in *Proceedings of The Genetic and Evolutionary Computation Conference*, 2022, pp. 620–629.

[266] M. Steinbach and P.-N. Tan, "kNN: k-nearest neighbors," in *The top ten algorithms in data mining*, Chapman and Hall/CRC, 2009, pp. 165–176.

[267] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.

[268] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, p. 21, 2013.

[269] J. Liu, C. Yang, Z. Lu, *et al.*, "Towards graph foundation models: A survey and beyond," *arXiv preprint arXiv:2310.11829*, 2023.

[270] N. Kooverjee, S. James, and T. Van Zyl, "Investigating transfer learning in graph neural networks," *Electronics*, vol. 11, no. 8, p. 1202, 2022.

Bibliography

Publications Related to the Dissertation

Journal Articles

A. Kostovska, J. Bogatinovski, S. Džeroski, D. Kocev, and P. Panov, "A catalogue with semantic annotations makes multilabel datasets FAIR," *Scientific Reports*, vol. 12, no. 1, p. 7267, 2022.

A. Kostovska, D. Vermetten, C. Doerr, S. Džeroski, P. Panov, and T. Eftimov, "OPTION: OPTImization Algorithm Benchmarking ONtology," *IEEE Transactions on Evolutionary Computation*, vol. 27, no. 6, pp. 1618–1632, 2023. DOI: 10.1109/TEVC.2022.3232844.

T. Eftimov, G. Petelin, G. Cenikj, *et al.*, "Less is more: Selecting the right benchmarking set of data for time series classification," *Expert Systems with Applications*, vol. 198, p. 116 871, 2022.

A. Kostovska, D. Vermetten, P. Korošec, S. Džeroski, C. Doerr, and T. Eftimov, "Using machine learning methods to assess module performance contribution in modular optimization frameworks," *Evolutionary Computation*, pp. 1–27, Aug. 2024, ISSN: 1063-6560. DOI: 10.1162/evco_a_00356.

Conference Papers

A. Kostovska, S. Džeroski, and P. Panov, "Semantic description of data mining datasets: An ontology-based annotation schema," in *Proceedings of International Conference on Discovery Science*, Springer, 2020, pp. 140–155.

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.

A. Kostovska, A. Jankovic, D. Vermetten, *et al.*, "Per-run algorithm selection with warm-starting using trajectory-based features," in *Proc. of Parallel Problem Solving from Nature (PPSN)*, Springer, 2022, pp. 46–60.

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 *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, ACM, 2022, pp. 648–656.

A. Kostovska, D. Vermetten, C. Doerr, S. Džeroski, P. Panov, and T. Eftimov, "OPTION: OPTImization Algorithm Benchmarking ONtology," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2021, pp. 239–240.

A. Kostovska, C. Doerr, S. Džeroski, D. Kocev, P. Panov, and T. Eftimov, "Explainable Model-specific Algorithm Selection for Multi-Label Classification," in *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2022, pp. 39–46.

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.

A. Kostovska, G. Cenikj, D. Vermetten, *et al.*, "PS-AAS: Portfolio Selection for Automated Algorithm Selection in Black-Box Optimization," in *International Conference on Automated Machine Learning*, PMLR, 2023, pp. 11–1.

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*, 2023, pp. 495–498.

A. Nikolikj, A. Kostovska, D. Vermetten, C. Doerr, and T. Eftimov, "Quantifying individual and joint module impact in modular optimization frameworks," *arXiv preprint arXiv:2405.11964*, 2024.

Publications Not Related to the Dissertation

Journal Articles

B. Stevanoski, A. Kostovska, P. Panov, and S. Džeroski, "Change detection and adaptation in multi-target regression on data streams," *Machine Learning*, pp. 1–38, 2024.

I. Dimitrovski, I. Kitanovski, P. Panov, A. Kostovska, N. Simidjievski, and D. Kocev, "AiTLAS: Artificial intelligence toolbox for earth observation," *Remote Sensing*, vol. 15, no. 9, p. 2343, 2023.

M. Petković, L. Lucas, J. Levatić, *et al.*, "Machine-learning ready data on the thermal power consumption of the Mars Express Spacecraft," *Scientific Data*, vol. 9, no. 1, p. 229, 2022.

Conference Papers

A. Kostovska, M. Petković, T. Stepišnik, *et al.*, "GalaxAI: Machine learning toolbox for interpretable analysis of spacecraft telemetry data," in *2021 IEEE 8th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, IEEE, 2021, pp. 44–52.

Biography

Ana Kostovska was born on 14.04.1995 in Strumica, North Macedonia. She finished her primary and secondary school in Strumica, North Macedonia. In 2013, she started her bachelor's at the Faculty of Computer Science and Engineering of Ss. Cyril and Methodius University in Skopje, Macedonia. During her undergraduate studies, she held a state scholarship for talented students awarded by the Ministry of Education and Science of North Macedonia.

In 2017, after successfully finishing her undergraduate studies with a GPA of 9.15, she started her master's studies in Information and Communication Technologies at the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. During her master's studies, she was awarded the Ad Futura scholarship from the Public Scholarship, Development, Disability, and Maintenance Fund of the Republic of Slovenia. In September 2019, she completed her master's studies with an average grade of 9.95 under the supervision of doc. dr. Panče Panov and co-supervision of Prof. dr. Sašo Džeroski.

In 2019, she started to work as a young researcher at the Department of Knowledge Technologies, Jožef Stefan Institute in Ljubljana, under the supervision of Prof. dr. Sašo Džeroski. In 2017, she enrolled in the PhD study program in Information and Communication Technologies at the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia under the supervision of doc. dr. Panče Panov and co-supervision of Prof. dr. Sašo Džeroski and doc. dr. Tome Eftimov. During her PhD studies, she received the SPECIES scholarship and conducted a three-month research visit at LIP6, Sorbonne University in Paris, under the supervision of dr. Carola Doerr.

Her research interests lie in the field of machine learning and knowledge representation and reasoning. Specifically, her work has focused on formalizing knowledge in a variety of domains, such as machine learning, process-based modeling, and black-box optimization, in the form of ontologies. Her goal is to improve the reusability and reproducibility of research resources and to develop new resources for knowledge representation and reasoning that can be applied to various domains. Additionally, Ana is actively interested in the application of machine learning techniques for algorithm selection and algorithm performance prediction tasks.