Algorithm Selection for Multi-Objective Optimization

by

ALEXANDRE DANIEL BORGES DE JESUS

ajesus@dei.uc.pt

A thesis submitted to the University of Coimbra in partial fulfillment of the requirements for the doctoral degree in Information Science and Technology, and to the University of Lille in partial fulfillment of the requirements for the doctoral degree in Computer Science, under a co-tutelle agreement between both institutions.

Prepared under the supervision of

LUÍS PAQUETE
Associate Professor
University of Coimbra

BILEL DERBEL
Associate Professor
University of Lille

ARNAUD LIEFOOGHE
Associate Professor
University of Lille

This thesis was financially supported by Portuguese Foundation for Science and Technology under scholarship SFRH/BD/132275/2017 co-funded by the European Social Fund and by the State Budget of the Portuguese Ministry of Education and Science. It was also partially supported by PICS project MOCO-SEARCH co-funded by the French National Center for Scientific Research and the Portuguese Foundation for Science and Technology.

This document was typeset with LaTeX and BibLaTeX, in conjunction with various other TeX packages that were graciously made freely available by their authors in the Comprehensive TeX Archive Network (CTAN) and that are part of the TeX Live distribution.

Abstract

Multi-objective optimization problems, which consider multiple objective functions to be optimized, can arise in many real-life scenarios, e.g., when trying to minimize both the cost and the time needed for traveling between two locations. In the last few decades, several algorithms have been proposed to solve multi-objective optimization problems. These algorithms can have very distinct behaviors, and their performance is often significantly affected by the problem instance to be solved, the time budget available, or the desirable solution quality. As such, which algorithm performs best often depends on the scenario that is being considered.

This gives rise to the algorithm selection problem, which is concerned with the automatic selection of the best algorithm for a given scenario. In this thesis, we investigate the case of automatically selecting the best multi-objective optimization algorithm to solve a previously unseen problem instance, taking into account that the available time budget and desirable solution quality may be uncertain, and are only known when selecting the algorithm. We make several contributions in this line.

First, we propose theoretical and empirical models to characterize the anytime performance of an algorithm, i.e., how solution quality improves over time, for previously unseen problem instances. Then, considering these models, we develop an offline selection methodology to select the best algorithm for a previously unseen problem instance given a utility function that describes the desirable time budget and solution quality. We also propose an online selection methodology that can swap between multi-objective branch and bound strategies to improve anytime performance. Lastly, we propose a scalarization technique and a branch and bound search strategy for multi-objective optimization problems to achieve a better anytime performance than previous approaches. Each contribution is backed by an experimental study on a multi-objective knapsack problem, and the results highlight the quality of the proposed models, selection methodologies, and algorithms.

Resumo

Problemas de otimização multi-objetivo, que consideram múltiplas funções objetivo a otimizar, podem surgir em diversos cenários reais, por exemplo, quando se quer minimizar tanto o custo como o tempo de uma viagem entre dois locais. Nas últimas décadas, vários algoritmos foram propostos para resolver problemas multi-objetivo. Estes algoritmos podem ter comportamentos distintos, e o seu desempenho é tipicamente afetado pela instância do problema a resolver, o tempo disponível para resolver o problema, e a qualidade da solução desejada. Como tal, qual o algoritmo que tem melhor desempenho depende do cenário em consideração.

Isto dá origem ao problema de seleção de algoritmos, que considera a escolha automática do algoritmo com melhor desempenho para um dado cenário. Nesta tese, investigamos a seleção automática do melhor algoritmo para resolver instâncias do problema nunca antes vistas, tendo em conta que o tempo disponível e a qualidade da solução desejada podem ser incertos, e apenas conhecidos aquando da seleção. Fazemos várias contribuições nesta direção.

Em primeiro lugar, propomos modelos teóricos e empíricos para caracterizar o desempenho de algoritmos *anytime*, ou seja, modelos que caracterizem a evolução da qualidade da solução devolvida pelo algoritmo ao longo do tempo, para instâncias do problema nunca antes vistas. Em segundo lugar, tendo em conta os modelos propostos, desenvolvemos uma metodologia de seleção *offline* para selecionar o melhor algoritmo para uma instância do problema nunca antes vista, dada uma função de utilidade que descreve o tempo disponível e a qualidade da solução desejada. Também propomos uma metodologia de seleção *online* capaz de mudar de estratégias *branch and bound* multi-objetivo de forma a melhorar o desempenho do algoritmo ao longo do tempo. Por fim, propomos uma técnica de escalarização e uma estratégia de *branch and bound* para problemas de otimização multi-objetivo para obter um melhor desempenho ao longo do tempo comparativamente a abordagens já existentes. Cada contribuição é acompanhada por um estudo experimental de um problema *knapsack* multi-objetivo, sendo que os resultados destacam a qualidade dos modelos, metodologias de seleção, e algoritmos propostos.

Résumé

Les problèmes d'optimisation multi-objectifs, pour lesquels plusieurs objectifs doivent être optimisés, peuvent survenir dans de nombreux scénarios réels, par exemple lorsque l'on essaie de minimiser à la fois le coût et le temps nécessaires pour se déplacer entre deux emplacements. Au cours des dernières décennies, de nombreux algorithmes ont été proposés afin de résoudre des problèmes d'optimisation multi-objectifs. Ces algorithmes peuvent avoir des comportements très distincts et leurs performances sont souvent affectées de manière significative par l'instance du problème à résoudre, le budget temps disponible pour la résolution, ou encore la qualité de solution souhaitée. Ainsi, l'algorithme qui fonctionne le mieux dépend souvent du scénario envisagé.

Cela donne lieu au problème de sélection d'algorithme, qui concerne la sélection automatique du meilleur algorithme pour un scénario donné. Dans cette thèse, nous étudions le cas de la sélection automatique du meilleur algorithme d'optimisation multi-objectifs pour résoudre une instance de problème non-rencontrée auparavant, en tenant compte du fait que le budget temps disponible et la qualité de solution souhaitée peuvent être incertains, et ne sont connus qu'à l'étape de la sélection de l'algorithme. Nous apportons plusieurs contributions dans cette voie.

Dans un premier temps, nous proposons des modèles théoriques et empiriques pour caractériser la performance "anytime" d'un algorithme, c'est-à-dire comment la qualité de solution s'améliore au fil du temps, pour des instances de problème non-rencontrées auparavant. Ensuite, en considérant ces modèles, nous développons une méthodologie de sélection hors ligne afin de sélectionner le meilleur algorithme étant donné une fonction d'utilité qui décrit le budget temps et la qualité de solution souhaités. Nous proposons également une méthodologie de sélection en ligne qui peut basculer entre des stratégies de branch and bound multi-objectifs pour améliorer les performances "anytime". Enfin, nous proposons une technique de scalarisation et une stratégie de branch and bound pour l'optimisation multi-objectifs afin d'obtenir une meilleure performance "anytime" que les approches précédentes. Chaque contribution est étayée par une étude expérimentale sur un problème de sac à dos multi-objectifs, et les résultats mettent en évidence la qualité des modèles, des méthodologies de sélection et des algorithmes proposés.

Acknowledgements

As I am working on the final details of this thesis, I cannot help but reflect on the countless hours, nights, discussions, gatherings, that have led me to this moment, and realize that I was fortunate to have so many people around to support me, both technically and emotionally, without whom this work would not have been possible.

I owe my first thanks to my supervisors, Luís Paquete, Arnaud Liefooghe, and Bilel Derbel, who were always there to guide me with their expertise and support me in this journey, who always treated me very kindly, and who helped me grow a great deal. To Bilel, thank you for all the help and advice, for making me feel a part of the team during my year in Lille, and for all the opportunities while I was there. To Arnaud, thank you for suggesting this interesting topic, and for making me feel welcome in Lille. Thank you also for always being there to answer any questions despite the distance most of the time, and thank you for always being available to discuss anything.

To Luís, I owe a great deal to you. Thank you for instilling in me the curiosity for optimization from the first day I step foot in one of your classrooms, nearly 12 years ago, and for continuously challenging me during my bachelor and masters at various stages, all of which eventually led me here. Thank you for your patience, for the countless discussions we had, for always being quick to give great feedback, for all the opportunities, and for all the pieces of advice, all of which were tremendously impactful and allowed me to grow as a researcher, as a teacher, and as a person.

I also owe a special thanks to all the people with whom I was fortunate to share an office in Coimbra and in Lille. In particular, thank you to Andreia Guerreiro, Geoffrey Pruvost, Gonçalo Lopes, João Macedo, Maryam Abbasi, Nicolas Berveglieri, Pedro Correia, Pedro Rodrigues, Stephan Helfrich, Tiago Carneiro. I am indebted to all of you for your friendship, for the great discussions and work environment, and for making this such a pleasant journey. A special thanks also to Carlos Fonseca, whose curiosity, rigor, and passion for research are contagious and have set a high standard for me.

Thank you to Luís Rodrigues and Luís Coimbra for inviting me to be a part of NEDUC, and thank you to all NEDUC members who work to make the University of Coimbra a better place for all its PhD students.

Thank you also to all my friends who have been there throughout this journey and throughout my life. Thank you for always being there to help build such great memories, and for always being there to offer a kind word and support when needed.

However, none of this would have been possible without the support of my loving family, my parents, my brother, and my aunt, who have always been there to support me, to provide for me, to advise me, to teach me, and who have always encouraged me to follow my passions.

And Fátima, my partner in everything, thank you for being my home, for always being by my side, and for always believing in me and encouraging me without hesitation. I am forever in your debt.

Lastly, to Coco, thank you for being a constant source of joy from the moment you entered my life.

Table of Contents

Chapter 1

Introduction

1.1 Motivations

Multi-Objective Optimization (MOO) deals with the study of problems that have more than one objective to optimize. Such problems may arise in many real-world applications. For example, when considering the problem of finding the optimal path between two cities, several objectives may be considered simultaneously, such as minimizing the traveling time, minimizing the monetary cost of the trip, and maximizing the number of attractions in the path.

One difficulty that commonly arises when tackling MOO problems is the conflicting nature of the objectives. This typically implies that there exists no single solution that optimizes all objectives simultaneously. Instead, there is a set of *efficient solutions*, namely the *efficient set*, that represent the optimal trade-offs between the objectives. A solution is said to be *efficient* if there is no other solution that is better or equal for all objectives, and strictly better for at least one objective. Going back to the previous example, the use of highways instead of national roads typically minimizes the time needed to travel between two cities. However, toll costs increase the overall cost of the trip. On the contrary, opting for national roads can minimize the total cost of the trip, but is likely to increase the travel time. As such, for a non-trivial trip, there may be a large number of efficient solutions that consider different stretches of highway and national roads.

Under the assumption that no preference regarding the objectives is known a priori, every efficient solution is potentially relevant to a Decision Maker (DM). Therefore, a common goal in MOO is to find the entire efficient set. Unfortunately, finding the entire set in a reasonable amount of time can be infeasible for many problems [15]. As such, it is often more useful to find a good approximation to the efficient set, namely an *approximation set*, within a reasonable amount of time. However, it is not always clear what is a reasonable amount of time or a good approximation set when designing an algorithm. For example, consider an algorithm for a real-time system that can take more or less time to respond to an event depending on the urgency of the situation.

Anytime algorithms [11, 78] return an approximation for any time budget, thus offering a trade-off between execution time and approximation quality. As a result, anytime algorithms are particularly suited for situations like the one described above, where the *anytime preferences* of the DM with respect to the desired approximation set quality or available time budget are uncertain. Interestingly, many commonly used MOO algorithms are inherently anytime since they keep and extend an *archive* of feasible solutions, which can be returned at any time.

Although many anytime MOO algorithms have been proposed over the years, it is often not clear which algorithm should be selected to solve a previously unseen problem instance. This selection should take into account the anytime preferences of the DM since they may impact the choice. For example, consider a real-time system that receives problem instances to solve. For some instances, the system needs to return a solution within a short amount of time, e.g., 1 second. For other instances, the system has more time, e.g., 60 seconds. For yet other instances, the exact time depends on external factors that are not yet fully established, but it is known that the algorithm will be interrupted between two time points, e.g., between 1 and 60 seconds. Depending on the time available to solve the problem instance, a different algorithm may be chosen.

The problem of automatically selecting an algorithm is commonly known as the Algorithm Selection Problem (ASP), which was introduced by Rice [67] and has gathered the attention of many researchers, on different areas, in recent years [41, 44, 69]. However, research for the ASP has mostly focused on selecting between single-objective optimization algorithms, and existing methodologies are not easily extendable to the multi-objective case [41]. Moreover, existing algorithm selection methodologies often consider that the anytime preferences of the DM are known in advance, i.e., when designing or training the selection methodology. However, in scenarios like the one described in the previous paragraph, anytime preferences are only known when performing the selection, or during the execution of the selected algorithm itself.

1.2 Goals & Scope

In this thesis, we are interested in studying the ASP for selecting between MOO algorithms to solve a previously unseen problem instance. We consider, two main perspectives for algorithm selection, *offline* and *online*. In the offline case, our goal is to select the best anytime algorithm to solve a previously unseen problem instance, before attempting to solve it. Moreover, we consider that the anytime preferences of the DM are only known when calling the algorithm selection methodology. In the online case, we want to select and change algorithms while solving a previously unseen problem instance. In this case, we consider that the anytime preferences of the DM are unknown, thus the goal is to optimize the quality of the returned approximation set over time, namely to improve *anytime performance*.

Algorithm selection methodologies often make use of *theoretical* or *empirical models* to predict the performance of algorithms on previously unseen instances. In this thesis, we consider the design of such models for predicting the anytime performance of MOO algorithms. We distinguish between theoretical and empirical models in the sense that latter are built using empirically collected data from previous runs of an algorithm on known problem instances, whereas the former rely only on the theoretical knowledge of how an algorithm operates.

We consider that the anytime performance of a MOO algorithm is characterized by a function that maps the time taken by an algorithm, to the quality of the approximation set that would be returned if the algorithm was interrupted at that time. For measuring time, we consider the CPU-time taken by the algorithm. However, since CPU-time depends on the empirical environment, we consider the number of iterations performed by the algorithm when designing theoretical models. For measuring the quality of an approximation set as a scalar value, we consider the *hypervolume indicator* [80], a commonly used quality indicator in MOO that measures the multi-dimensional area dominated by the image of an approximation set in the objective space, with respect to a given reference point.

As such, the main research questions considered for this thesis are the following:

R1. Can we design theoretical models to predict the anytime performance of particular MOO algorithms for previously unseen instances with respect to the number of iterations performed by the algorithm, taking into account the theoretical behavior of the algorithm?

R2. Can we build empirical models to predict the anytime performance of MOO algorithms for previously unseen instances from a set of training instances?

R3. Can we design an offline algorithm selection methodology to select between MOO algorithms for solving a previously unseen instance using the previously designed empirical models of anytime performance?

R4. Can we design an online algorithm selection methodology to select and swap between MOO algorithms such that anytime performance is optimized?

1.3 Contributions

The main contributions of this thesis, which directly connect to the research questions presented above, are as follows:

C1. We propose, and empirically analyze, two variants of theoretical model of anytime performance for bi-objective optimization algorithms that find, at each iteration, a solution maximizing the accumulated hypervolume. Anytime performance is considered in terms of the number of iterations performed by the algorithm, and the hypervolume of the archive of collected solutions. The first variant is given by an analytical methodology that while simple, requires several assumptions. In particular, it assumes that the non-dominated set can be approximated by a piecewise linear function with two symmetric segments and no discontinuities. Furthermore, it requires that the objective space is scaled down such that the non-dominated set is contained in the unit square $[0, 1]^2$, and that the reference point for the hypervolume is set to the origin. The second variant is given by an algorithm that generalizes the ideas of the first variant to require less assumptions. In particular, this variant only requires that the non-dominated set can be approximated by a piecewise linear function.

C2. We propose three empirical models to predict the anytime performance of a MOO algorithm for a previously unseen problem instance. For building these models we assume that there is a set of training problem instances that can be solved by the algorithm. These models differ in their assumptions, and in their predictive power. In particular, the first model has no other assumptions besides having a set of training instances that can be characterized by a set of instance features, and having collected anytime performance data for the algorithms on those instances. However, it can only predict anytime performance for which anytime performance data has been collected, e.g., if anytime performance data was only collected up to a CPU-time of 10 seconds, it cannot predict anytime performance for a CPU-time of 20 seconds. On the other hand, the second model can predict beyond the collected anytime performance data used for training, but assumes that the anytime performance of an algorithm can be characterized by a (non-linear) mixed effects model. The third model, assumes that a theoretical model of anytime performance that maps the number of iterations to the archive quality exists for the current algorithm. Its predictive accuracy depends on the quality of this theoretical model, and on the accuracy of predicting how much computational time each iteration takes. Lastly, we perform an empirical study on the first model.

C3. We propose, and empirically analyze, an offline algorithm selection framework to select the best anytime MOO algorithm to solve a previously unseen problem instance, under the assumption that the anytime preferences of the decision maker are only (partially) known when performing the selection and can be characterized by a utility function with respect to time and quality. This framework considers empirical models of anytime performance, such as the ones described in contribution C2, to predict the anytime performance of the algorithms on the previously unseen problem instance. Then, a scalar measure of anytime performance is computed for each algorithm, which takes into account the predicted anytime performance and the utility function describing the anytime preferences of the DM. Finally, the algorithm that optimizes this measure of anytime performance is selected.

C4. We propose, and empirically analyze, an online algorithm selection methodology to automatically select and swap between MOO algorithms in order to optimize anytime performance. In particular, this methodology selects between different branch-and-bound approaches depending on the active nodes and archive of collected solutions.

The contributions above also led to new anytime MOO approaches:

C5. We propose an ε-constraint scalarization approach for bi-objective optimization problems guided by the theoretical model of anytime performance from contribution C1.

C6. We propose an indicator-based branch-and-bound framework for MOO that follows a best-first search strategy. This framework considers the use of a binary quality indicator to select the best node to be explored at each iteration. This approach is considered in the context of the online selection methodology of contribution C4.

1.4 Publications & Software

The following publications, in chronological order, are a direct result from the work carried out for this thesis:

P1. A. D. Jesus, L. Paquete, and A. Liefooghe. "A Model of Anytime Algorithm Performance for Biobjective Optimization Problems". In: *Proceedings LeGO - 14th International Global Optimization Workshop*. LeGO 2018. Vol. 2070. AIP Conference Proceedings. AIP, 2019, p. 020049. DOI: 10.1063/1.5090016 [39]

In this work, presented at the 14th International Workshop on Global Optimization (LeGO) organized at the Leiden University, we propose and empirically analyze the analytical theoretical model of contribution C1.

P2. A. D. Jesus, L. Paquete, and A. Liefooghe. "A Model of Anytime Algorithm Performance for Bi-Objective Optimization". In: *Journal of Global Optimization* 79 (2020), pp. 329–350. DOI: 10.1007/s10898-020-00909-9 [38]

In this work, we propose and empirically analyze the algorithmic theoretical model of contribution C1, as well as, the ε-constraint approach of contribution C5.

P3. A. D. Jesus, A. Liefooghe, B. Derbel, and L. Paquete. "Algorithm Selection of Anytime Algorithms". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO 2020. Association for Computing Machinery, 2020, pp. 850–858. DOI: 10.1145/3377930.3390185 [36]

In this work, we propose and empirically analyze the first empirical model of contribution C2, as well as, the offline algorithm selection framework of contribution C3.

P4. A. D. Jesus, L. Paquete, B. Derbel, and A. Liefooghe. "On the Design and Anytime Performance of Indicator-based Branch and Bound for Multi-objective Combinatorial Optimization". In: *Proceedings of the 2021 Genetic and Evolutionary Computation Conference*. GECCO 2021. Association for Computing Machinery, 2021, pp. 234–242. DOI: 10.1145/3449639.3459360 [37]

In this work, we propose and empirically analyze the indicator-based branch-and-bound of contribution C6.

P5. A. D. Jesus, L. Paquete, A. Liefooghe, and B. Derbel. "Techniques to Analyze the Anytime Behavior of Algorithms for Multi-Objective Optimization". 31st European Conference on Operational Research (EURO 2021). 2021 [40]

In this talk, we present an `R` package to measure, and visually analyze, the anytime performance of anytime MOO algorithms.

The following work, is not directly related to this thesis, but analyzes techniques that were used in the implementation of the algorithms for this thesis:

P6. D. M. Dias, A. D. Jesus, and L. Paquete. "A Software Library for Archiving Nondominated Points". In: *Proceedings of the 2021 Genetic and Evolutionary Computation Conference Companion*. GECCO 2021. Association for Computing Machinery, 2021, pp. 53–54. DOI: `10.1145/3449726.3462737` [13]

In this work we present, and empirically analyze, a `C++` library, available at [14], to perform operations on sets of points in the objective space. Currently, it includes functions to filter the non-dominated points of a set, and to update sets of non-dominated points.

The following software resulted from the development of this thesis and has been made publicly available:

S1. A. D. Jesus. *mooutils*. Version v0.1.0. Zenodo, 2022. DOI: `10.5281/zenodo.6855879` [35]

A `C++` library of multi-objective optimization utilities, including: functions to compare solutions in terms of dominance and lexicographic orders, data structures to keep mutually non-dominated solutions, data structures and functions for quality indicators such as the hypervolume, different solution queues for search algorithms that keep and process (partial) solutions in specific orders.

S2. A. D. Jesus. *mobkp*. Version v0.1.1. Zenodo, 2022. DOI: `10.5281/zenodo.6857821` [33]

A `C++` library for the Multi-Objective Binary Knapsack Problem (MOBKP) containing several state-of-the-art algorithms, including the algorithms of contributions C5 and C6, as well as, a novel data generation procedure described in contribution C2.

S3. A. D. Jesus. *moco_abm*. Version v0.2.0. Zenodo, 2019. DOI: `10.5281/zenodo.3548869` [34]

A `Rust` implementation of the algorithmic anytime performance model of contribution C1.

S4. A. D. Jesus. *apm*. Version v0.1.1. Zenodo, 2022. DOI: `10.5281/zenodo.6857541` [31]

A `C++` library containing the theoretical models of anytime performance of contribution C1.

S5. A. D. Jesus. *anytime*. Version v0.0.2. Zenodo, 2022. DOI: `10.5281/zenodo.6856120` [30]

An `R` library for the analysis of anytime performance presented in publication P5.

Lastly, the code for the experiments carried out in this thesis is publicly available at [32].

1.5 Outline

The remainder of this thesis is organized as follows. In Chapter 2, we provide the general concepts and definitions related to MOO, anytime performance and algorithm selection. In Chapter 3, we present the theoretical models and experimental analysis of contribution C1, as well as, the ε-constraint algorithm of contribution C5. In Chapter 4, we present the empirical models and experimental analysis of contribution C2. In Chapter 5, we present the offline algorithm selection framework and experimental analysis of contribution C3. In Chapter 6, we give the online selection methodology and experimental analysis of contribution C4, as well as the branch-and-bound algorithm of contribution C6. In Chapter 7, we summarize and discuss the main findings of this work, and possible directions for future work.

Chapter 2

Background

In this chapter, we provide the technical background for this thesis. It is organized as follows. In Section 2.1, we give the concepts and notations related to MOO. In Section 2.2, we present the quality indicators that are used to measure the quality of an approximation set, and discuss their properties. In Section 2.3, we introduce a benchmark MOO problem and present several approaches to solve it. In Section 2.4, we provide the concepts related to anytime algorithms and anytime performance. Lastly, in Section 2.5, we present the ASP and discuss some techniques that have been employed to solve this problem.

2.1 Multi-Objective Optimization

A MOO problem is characterized by having several, typically conflicting, objectives to be optimized. Assuming, without loss of generality, that all objectives are to be maximized, a MOO problem with m objectives is defined as:

$$\operatorname*{argmax}_{x \in \mathcal{X}} f(x) = (f_1(x), f_2(x), \ldots, f_m(x)) \tag{2.1}$$

where $\mathcal{X}$ denotes the set of feasible solutions, and $f_i(x) \to \mathbb{R}$ denotes the i-th objective function. The space containing $\mathcal{X}$ is called the *decision space*. If the solutions follow a combinatorial structure we say that the problem is a Multi-Objective Combinatorial Optimization (MOCO) problem. The image $f(x)$ of a solution $x \in \mathcal{X}$ is called an *objective vector*. The set of all objective vectors is denoted by $\mathcal{Y} = \{f(x) : x \in \mathcal{X}\}$. The space containing $\mathcal{Y}$ is called the *objective space*.

To compare any two points in the objective space we consider the following *binary dominance relations*.

Definition 2.1 (Binary dominance relations)**.** Let $y^1, y^2 \in \mathbb{R}^m$. We say that:

- $y^1 \geq y^2$ (y^1 *weakly dominates* y^2) iff $y^1_i \geq y^2_i$ for all $i \in \{1, \ldots, m\}$.
- $y^1 \succ y^2$ (y^1 *dominates* y^2) iff $y^1 \geq y^2$ and $y^1 \neq y^2$.
- $y^1 > y^2$ (y^1 *strictly dominates* y^2) iff $y^1_i > y^2_i$ for all $i \in \{1, \ldots, m\}$.
- y^1 and y^2 are *mutually non-dominated* iff $y^1 \not\succ y^2$ and $y^2 \not\succ y^1$.

There is often no unique solution that simultaneously optimizes all the objectives in a MOO problem. Instead, there are several *efficient solutions* that represent the trade-offs between the objectives.

Definition 2.2 (Efficiency and non-dominance)**.** A solution $x \in \mathcal{X}$ is *efficient*, and its objective vector $f(x)$ is *non-dominated*, iff there is no other solution $x' \in \mathcal{X}$ such that $f(x') \succ f(x)$ holds. The set of all efficient solutions is called the *efficient set* and denoted by $\mathcal{X}_E$. The image of the efficient set in the objective space, $\mathcal{Y}_N = \{f(x) : x \in \mathcal{X}_E\}$, is called the *non-dominated set*.

In MOCO it is often useful to distinguish between *supported* and *non-supported* solutions [16].

Definition 2.3 (Supported solutions)**.** An efficient solution $x \in \mathcal{X}_E$ is said to be *supported* if it is also a solution to the weighted sum problem

$$\underset{x \in \mathcal{X}}{\operatorname{argmax}} \; \lambda f(x) \tag{2.2}$$

where $\lambda \in \mathbb{R}^m_{>0}$ denotes a vector of positive weights. Otherwise, x is said to be *non-supported*. The image in the objective space of a (non-)supported solution is called a *(non-)supported objective vector*. The sets of all supported solutions and objective vectors are denoted by $\mathcal{X}_S$ and $\mathcal{Y}_S$ respectively.

Definition 2.4 (Extreme supported solutions)**.** A solution $x \in \mathcal{X}_E$ is said to be an *extreme supported solution* if it is a supported solution and its objective vector $f(x)$ is an extreme point of the convex hull of $\mathcal{Y}_N$. The image of an extreme supported solution is denoted an *extreme supported objective vector*. The sets of all extreme supported solutions and objective vectors are denoted by $\mathcal{X}_{ES}$ and $\mathcal{Y}_{ES}$, respectively.

We also consider the following *lexicographical relations*.

Definition 2.5 (Lexicographical relations)**.** Let $y^1, y^2 \in \mathbb{R}^m$. We say that:

- $y^1 >_{\text{lex}} y^2$ (y^1 is *lexicographically greater than* y^2) iff $y^1_j > y^2_j$ subject to $j = \min\{i : y^1_i \neq y^2_i\}$.

- $y^1 \geq_{\text{lex}} y^2$ (y^1 is *lexicographically greater than or equal to* y^2) iff $y^1 = y^2$ or $y^1 >_{\text{lex}} y^2$.

The problem of finding the *lexicographically optimal solution*, assuming, without loss of generality, maximizing objectives, is denoted as:

$$\operatorname*{arglexmax}_{x \in \mathcal{X}} \ (f_1(x), \ldots, f_m(x)) \tag{2.3}$$

Note that every lexicographically optimal solution is also efficient [16].

In the following, we consider several definitions related to sets of solutions or points in the objective space. First, we consider the definition of a *minimal set*.

Definition 2.6 (Minimal set)**.** Let $X^1, X^2 \subseteq \mathcal{X}$ denote two sets of feasible solutions. We say that X^1 is a *minimal set* of X^2 iff $X^1 \subseteq X^2$ and for all $x^2 \in X^2$ there exists $x^1 \in X^1$ such that $f(x^1) = f(x^2)$.

Lastly, to compare between two sets of points in the objective space we define the following *binary set dominance relations*.

Definition 2.7 (Binary set dominance relations)**.** Let $Y^1, Y^2 \subseteq \mathbb{R}^m$ denote two sets of points in the objective space. We say that:

- $Y^1 \geq Y^2$ (Y^1 *weakly dominates* Y^2) iff for all $y^2 \in Y^2$ there exists $y^1 \in Y^1$ such that $y^1 \geq y^2$.

- $Y^1 > Y^2$ (Y^1 *dominates* Y^2) iff $Y^1 \geq Y^2$ and $Y^2 \ngeq Y^1$.

- $Y^1 \succ Y^2$ (Y^1 *strictly dominates* Y^2) iff for all $y^2 \in Y^2$ there exists $y^1 \in Y^1$ such that $y^1 > y^2$.

- Y^1 and Y^2 are *mutually non-dominated* iff $Y^1 \ngeq Y^2$ and $Y^2 \ngeq Y^1$.

2.2 Quality Indicators

The need to quantify the quality of a set as a scalar value often arises when comparing sets, e.g., to determine which of two mutually non-dominated sets is better. To this end, *quality indicators* have been proposed in the literature [43, 81]. We consider the following two classes of quality indicators.

Definition 2.8 (Unary quality indicator)**.** Let $A \subset \mathbb{R}^m$ be a set of points in the objective space. A *unary quality indicator* is a function $I(A) \to \mathbb{R}$ that maps the set to a real value.

Definition 2.9 (Binary quality indicator)**.** Let $A, B \subset \mathbb{R}^m$ be two sets of points in the objective space. A *binary quality indicator* is a function $I(A, B) \rightarrow \mathbb{R}$ that assigns a real value to set A with respect to set B.

In the following, we describe one unary and two binary quality indicators and their properties. These indicators were chosen for this thesis since they are commonly used in MOO, and because their order-preserving properties are of particular interest for the analysis of performance of anytime algorithms.

2.2.1 Hypervolume Indicator

A frequently used unary quality indicator is the *hypervolume indicator* [80], or simply hypervolume, which corresponds to the multi-dimensional area dominated by a set of points in the objective space, with respect to a reference point. Formally, for a set $A \subset \mathbb{R}^m$ and a reference point $r \in \mathbb{R}^m$, the hypervolume is defined by:

$$I_H(A) = \Lambda\left(\{q \in \mathbb{R}^m \mid \exists\, a \in A \colon r \geq q \geq a\}\right) \tag{2.4}$$

where Λ denotes the Lebesgue measure.

One important property of the hypervolume indicator is that it is strictly order-preserving with respect to the binary dominance relations introduced in Section 2.1 under a mild assumption [81]. In particular, let $A, B \subset \mathbb{R}^m$ denote two sets of points in the objective space such that every point in A and B strictly dominates the reference point $r \in \mathbb{R}^m$. Then, $A \succeq B$ implies that $I_H(A) \geq I_H(B)$, and $A \succ B$ implies that $I_H(A) > I_H(B)$.

This property gives a relevant implication for our work. In particular, if an anytime algorithm iteratively adds solutions to an archive set, then the hypervolume for that archive will not decrease. If the objective vector of the solution added to the archive is not weakly dominated by any other point in the archive, and if it strictly dominates the reference point, then the hypervolume for the archive will increase. Moreover, among all subsets of feasible objective vectors, the hypervolume is maximal for any set that contains the non-dominated set.

Another relevant property is that the hypervolume indicator is (strictly) order-preserving with respect to a (strictly) order-preserving scaling of the objectives [43]. This means that we can, for example, normalize the objectives without affecting the order of the approximation sets given by the hypervolume indicator. However, note that the order of the sets given by the hypervolume is affected by the setting of the reference point.

2.2.2 Binary Hypervolume Indicator

A binary quality indicator related to the hypervolume is the *binary hypervolume indicator* [79], or simply binary hypervolume, which corresponds to the multi-dimensional area dominated by one set but not by another, that is, given two sets $A, B \subset \mathbb{R}^m$, the binary hypervolume is given by:

$$I_H(A, B) = I_H(A \cup B) - I_H(B) \tag{2.5}$$

A relevant property of this indicator for this work, in particular for the indicator-based branch-and-bound approach described in Section 6.1, is that, under some mild assumptions, the binary hypervolume is (strictly) order-preserving when fixing the second parameter.

Proposition 2.1. Let $A, B, C \subset \mathbb{R}^m$ denote three sets of points in the objective space such that every point in A, B, and C, strictly dominates the reference point $r \in \mathbb{R}^m$, and A dominates C. Under these assumptions, the binary hypervolume is order-preserving for a fixed reference set R, that is:

$$A \geq B \implies I_H(A, C) \geq I_H(B, C) \tag{2.6}$$

Proof. Since $A \geq B$ and $A > C$ then it holds that:

$$A \cup C \geq B \cup C \tag{2.7}$$

Then, since all points in A, B, and C, strictly dominate the reference point and because the hypervolume is order-preserving, it holds that:

$$A \cup C \geq B \cup C \implies I_H(A \cup C) \geq I_H(B \cup C) \tag{2.8}$$

Lastly:

$$\begin{align} I_H(A \cup C) \geq I_H(B \cup C) \implies \tag{2.9}\\ I_H(A \cup C) - I_H(C) \geq I_H(B \cup C) - I_H(C) \implies \tag{2.10}\\ I_H(A, C) \geq I_H(B, C) \tag{2.11} \end{align}$$

□

Under the same assumptions, it similarly holds that the relation $A > B$ implies $I_H(A, C) > I_H(B, C)$.

It can similarly be shown that when fixing the first parameter, the binary hypervolume indicator is order-reversing with respect to the dominance of the second parameter, which is relevant to improve the performance of the branch-and-bound approach proposed in Section 6.1.

Proposition 2.2. Let $A, B, C \subset \mathbb{R}^m$ denote three sets of points in the objective space. Then binary hypervolume is order-reversing for a fixed first parameter, that is:

$$B \geq C \implies I_H(A, B) \leq I_H(A, C) \tag{2.12}$$

Proof. By definition:

$$I_H(A, B) \leq I_H(A, C) \implies \tag{2.13}$$

$$I_H(A \cup B) - I_H(B) \leq I_H(A \cup C) - I_H(C) \implies \tag{2.14}$$

$$I_H(A \cup B) - I_H(A \cup C) \leq I_H(B) - I_H(C) \tag{2.15}$$

Since $B \geq C$, this is equal to:

$$I_H(A \cup B \cup C) - I_H(A \cup C) \leq I_H(B \cup C) - I_H(C) \tag{2.16}$$

which, given the order-preserving property of the unary hypervolume, is equivalent to:

$$I_H(B \cup C) - I_H(C) - \epsilon \leq I_H(B \cup C) - I_H(C) \tag{2.17}$$

where $\epsilon \geq 0$ corresponds to the intersection of the multi-dimensional area that is covered by both A and B, but not by C. □

Note that by fixing one of the parameters, we could define the indicator as unary. However, we will not be setting a fixed parameter for the whole duration of an algorithm. Instead, we will perform some comparisons with one of the two parameters fixed at specific stages of the algorithm which is why these properties are relevant, but otherwise we keep changing both parameters as the algorithm progresses.

2.2.3 The ε-indicator

Another commonly used binary quality indicator is the *ε-indicator* [81]. For two sets $A, B \subset \mathbb{R}^m_{>0}$ the (multiplicative) ε-indicator corresponds to the smallest factor that can be applied to all points in A such that it weakly dominates B, formally:

$$I_\varepsilon(A, B) = \max_{r \in B} \min_{a \in A} \max_{i \in \{1,\ldots,m\}} r_i / a_i \tag{2.18}$$

where m is the number of objectives.

Like in the previous section, a relevant property for this indicator is that it is order-reversing when fixing the second parameter.

Proposition 2.3. Let $A, B, C \subset \mathbb{R}^m_{>0}$ denote three sets of points in the objective space. Then, the ε-indicator is order-reversing when fixing the second parameter, that is:

$$A \geq B \implies I_\varepsilon(A, C) \leq I_\varepsilon(B, C) \tag{2.19}$$

Proof. If $A \geq B$ then it holds that:

$$\forall b \in B\ \exists a \in A\ \forall i \in \{1, \ldots, m\}\ a_i \geq b_i \tag{2.20}$$

Then, it holds that:

$$\forall c \in C\ \forall b \in B\ \exists a \in A\ \forall i \in \{1, \ldots, m\}\ c_i/a_i \leq c_i/b_i \tag{2.21}$$

which implies that $I_\varepsilon(A, C) \leq I_\varepsilon(B, C)$. □

It also holds that the ε-indicator is order-preserving when fixing the first parameter.

Proposition 2.4. Let $A, B, C \subset \mathbb{R}^m_{>0}$ denote three sets of points in the objective space. Then, the ε-indicator is order-preserving when fixing the first parameter, that is:

$$B \geq C \implies I_\varepsilon(A, B) \geq I_\varepsilon(A, C) \tag{2.22}$$

Proof. If $B \geq C$ then it holds that:

$$\forall c \in C\ \exists b \in B\ \forall i \in \{1, \ldots, m\}\ b_i \geq c_i \tag{2.23}$$

Then, it holds that:

$$\forall a \in A\ \forall c \in C\ \exists b \in B\ \forall i \in \{1, \ldots, m\}\ b_i/a_i \geq c_i/a_i \tag{2.24}$$

which, from the definition of the ε-indicator, implies that $I_\varepsilon(A, B) \geq I_\varepsilon(A, C)$. □

2.3 Solving the Multi-Objective Binary Knapsack Problem

Throughout this thesis, we will use the MOBKP as a benchmark problem in our experimental studies. This problem has been commonly used as a benchmark problem in many MOCO studies and many algorithms have been proposed to solve it, which makes it an interesting problem for algorithm selection and anytime performance modeling. In the following, we introduce the MOBKP, and discuss several algorithmic approaches that have been proposed to solve it and which we will consider in this thesis.

Given a set of n items, each of which with an associated weight and $m > 1$ values, the MOBKP is the problem of finding the subset of items that maximizes the sum for each of the m values, such that the sum of the weights of the chosen items does not exceed a given capacity $W \in \mathbb{R}_{>0}$. More formally, it can be defined as follows:

$$\begin{aligned} \underset{x \in \{0,1\}^n}{\operatorname{argmax}} \; f(x) &= \left(f_1(x) = \sum_{i=1}^{n} v_i^1 x_i, \dots, f_m(x) = \sum_{i=1}^{n} v_i^m x_i \right) \\ \text{s.t.} \; \sum_{i=1}^{n} w_i x_i &\leq W \end{aligned} \tag{MOBKP}$$

where $v_i^j \in \mathbb{R}_{>0}$, $i \in \{1, \dots, n\}$, $j \in \{1, \dots, m\}$, denotes the value for item i and objective j, and $w_i \in \mathbb{R}^n_{>0}$ denotes the weight for item $i \in \{1, \dots, n\}$. The binary variable x_i denotes whether or not an item is chosen for the knapsack, $x_i = 1$ or $x_i = 0$ respectively.

When solving a MOCO problem the goal is typically to find the efficient set using an *exact approach*. However, for large instances, finding the efficient set in a reasonable amount of time may not be feasible. In such cases, the goal is to find a set of feasible solutions that gives a good approximation in a reasonable amount of time using an *heuristic approach*. In the following sections we describe several approaches for the MOBKP that are used throughout this thesis. A library containing implementations for these approaches is available at [33]. Note that many of these approaches can be adapted to other problems as well.

2.3.1 Scalarization Techniques

Scalarization techniques transform a MOO problem into a series of scalarized single-objective problems [16].

Dichotomic Weighted Sum

One commonly used scalarization technique for the bi-objective case ($m = 2$) is the Dichotomic Weighted Sum (DWS) method by Aneja and Nair [1]. This approach works by performing a dichotomic search on the weight values for a weighted-sum scalarization given by:

$$\begin{aligned} \underset{x \in \{0,1\}^n}{\operatorname{argmax}} \; & \lambda_1 f_1(x) + \lambda_2 f_2(x) \\ \text{s.t.} \; & \sum_{i=1}^{n} w_i x_i \leq W \end{aligned} \tag{2.25}$$

where $\lambda_1, \lambda_2 \in \mathbb{R}_{>0}$ represent the weight given to each objective. Note that, this scalarized problem corresponds to a single-objective binary knapsack problem, which can be solved by using an appropriate algorithm. The DWS method can be described as follows [1, 19]:

- Step 1: Compute lexicographical optimal solutions x^1 and x^2 for problems $\text{arglexmax}_{x \in \mathcal{X}}\ (f_1(x), f_2(x))$ and $\text{arglexmax}_{x \in \mathcal{X}}\ (f_2(x), f_1(x))$, respectively. Let $Q = \{(1, 2)\}$, $R = \{x^1, x^2\}$, and $k = 3$.

- Step 2: If Q is empty, stop the algorithm and return R. Otherwise, choose an element $(r, s) \in Q$ and let $Q = Q \setminus \{(r, s)\}$.

- Step 3: Let $\lambda_1 = f_2(x^s) - f_2(x^r)$, $\lambda_2 = f_1(x^r) - f_1(x^s)$. Let x^k be the optimal solution for the scalarized problem of Equation (2.25) with λ_1 and λ_2.

- Step 4: If $f(x^k) \neq f(x^r)$ and $f(x^k) \neq f(x^s)$, then let $R = R \cup \{x^k\}$, $S = S \cup \{(r, k), (k, s)\}$, $k = k + 1$. Go to Step 2.

At the end of the algorithm above the output set R is a subset of $\mathcal{X}_S$. If there are multiple optima at step 3 and the algorithm chooses one that maximizes $f_1(x)$ [5], then the output set R is also a minimal set of $\mathcal{X}_{ES}$ [1]. Since R cannot be guaranteed to match the efficient set, we consider this to be a heuristic approach. A generalization of this dichotomic scheme for more than two objectives is given by Przybylski et al. [66].

ε-constraint approaches

Another class of commonly used scalarization approaches are ε-constraint approaches. These approaches solve a sequence of scalarized problems where all objectives but one are turned into constraints. As an example, a scalarized ε-constraint problem for the MOBKP, where all objectives except the first one are turned into constraints, is defined as:

$$\begin{aligned} \underset{x \in \{0,1\}^n}{\text{argmax}}\ & f_1(x) \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i \leq W \\ & f_2(x) > \varepsilon_2 \\ & \dots \\ & f_m(x) > \varepsilon_m \end{aligned} \tag{2.26}$$

where ε_j denotes the constraint value for objective $j \in \{2, \dots, m\}$. We remark that the image of a solution to this problem can be dominated, but not strictly dominated, by an objective vector in the non-dominated set $\mathcal{Y}_N$ [16].

A commonly used ε-constraint method for two objectives is given by Srinivasan and Thompson [70]. This method starts by solving a scalarized problem with an arbitrarily small constraint ε_2 on the second objective. Then, at each iteration, given an optimal solution x^* to the scalarized problem in the previous iteration, it sets $\varepsilon_2 = f_2(x^*)$ as the constraint value for the new scalarized problem. A generalization of this idea for two or more objectives is the Adaptive ε-constraint (AEPS) method given by Laumanns et al. [45]. This approach works by partitioning the whole objective space into an hypergrid with respect to the objective functions f_2 to f_m based on the efficient solutions already identified. For each grid cell, up to m scalarized problems are solved to find an efficient solution. If an efficient solution is found, then the hypergrid is updated based on the objective values of that solution, otherwise the region for that grid cell is added to a set of already searched regions. The algorithm continues until all grid cells have been explored. We remark that, if the solver for the scalarized ε-constraint problems returns all optimal solutions for each problem, then this algorithm can find the efficient set. However, if the algorithm can only return one optimal solution for each problem, as is often the case, then the algorithm finds a minimal set of the efficient set.

2.3.2 Branch-and-Bound

Branch-and-Bound (BB) approaches implicitly visit all feasible solutions, which are added to an archive as the algorithm progresses, by recursively dividing the decision space such that each subdivision involves a subproblem. This recursive division is commonly represented by a search tree over the decision variables such that each *node* corresponds to a *partial solution* with some decision variables being fixed, and a *leaf* corresponds to a complete solution. The subproblem of each node gives rise to a lower and upper bound set. A *lower bound set* is a non-empty set of feasible solutions to the subproblem. An *upper bound set* is a non-empty set of pairwise mutually non-dominated points in the objective space that weakly dominates the non-dominated set of the subproblem [17, 65].

In the following we present two standard BB frameworks. The first, given in Algorithm 1, corresponds to an *eager* BB framework where the lower and upper bound sets of the nodes are computed as soon as that node is found. The second, given in Algorithm 2, corresponds to a *lazy* BB framework where the lower and upper bound sets are computed when the node is selected for exploration. For both approaches we start by defining a set of active nodes Q initially containing the *root* node r of the search tree. For the MOBKP, this node is the partial solution with no fixed variables. Then, archive S, which is used to keep an up to date set of pairwise mutually non-dominated solutions, is initialized. In the case of the eager framework the archive is initialized with the lower bound set of the root node. For the case of the lazy framework the archive is initially empty.

Then, at each iteration, a node is selected from the set of active nodes Q to be explored. In the eager framework, this exploration consists of computing the branching nodes, and adding them to the set of active nodes Q if the upper bound set of the branching node is not strictly dominated by the archive S. Otherwise, the branching node can be discarded. If the branching node is not discarded, then the lower bound set of the branching node is also used to update the archive set. In the lazy framework, we start by evaluating whether or not the upper bound set of the current node is strictly dominated by the archive S. If this is true, then that node can be discarded. Otherwise, the lower bound set of the node is used to update the archive, and the branching nodes are added to the set of active nodes Q.

Algorithm 1: Eager Branch-and-Bound Framework

```
Input  : Root node $r$
Output : Solution set $s$
S ← GetLowerBound($r$)
$Q \leftarrow \{r\}$
while $Q \neq \emptyset$ do
    node ← SelectNode($Q$)
    $Q \leftarrow Q \setminus \{$node$\}$
    for branch ∈ GetBranches(node) do
        if $S \nsucc$ GetUpperBound(branch) then
            $S \leftarrow S \cup$ GetLowerBound(branch)
            $Q \leftarrow Q \cup \{$branch$\}$
        end
    end
end
return $S$
```

We remark that most approaches in the literature, e.g., [52, 65, 75], follow an eager approach. Moreover, we note that the computation of a lower bound set is not strictly required since the leaf nodes correspond to all complete feasible solutions. However, by using a lower bound set, the algorithm can add more complete solutions to the archive earlier, which can lead to more branches being discarded earlier. This can often outweigh the computational cost of finding the lower bound set and leads to better anytime performance, and execution time.

In the following sections we discuss which strategies can be employed by BB approaches to: i) select the next node to be explored; ii) compute the

Algorithm 2: Lazy Branch-and-Bound Framework

```
Input  : Root node $r$
Output : Solution set $s$
1  $S \leftarrow \emptyset$
2  $Q \leftarrow \{r\}$
3  while $Q \neq \emptyset$ do
4      node $\leftarrow$ SelectNode($Q$)
5      $Q \leftarrow Q \setminus \{$node$\}$
6      if $S \not\succ$ GetUpperBound(node) then
7          $S \leftarrow S \cup$ GetLowerBound(node)
8          for branch $\in$ GetBranches(node) do
9              $Q \leftarrow Q \cup \{$branch$\}$
10         end
11     end
12 end
13 return $S$
```

branching nodes; and iii) compute the lower and upper bound sets.

Node Selection

Two commonly used search strategies for selecting the next node to be explored are Depth First Search (DFS) [52, 75] and Breadth First Search (BFS) [75], which can be implemented in Algorithms 1 and 2 by using a last-in first-out queue and a first-in first-out queue for the set of active nodes Q, respectively. We remark, that the BFS strategy often suffers from memory issues since it may need to keep a large amount of nodes in memory. On the other hand, the DFS strategy can easily get stuck deep in the tree by attempting to explore nodes that are already dominated by those in the archive.

Therefore, a third strategy, namely Best First Search (BeFS), is to select the most promising node in the set of active nodes Q according to some heuristic. In single-objective optimization this often corresponds to finding the node with the best, e.g., largest in the case of a maximizing objective, upper bound, since it suggests that searching in that direction will lead to a solution with a better objective value. For MOO this is not as straightforward since we are often dealing with incomparable upper bound sets. This issue is further discussed in Chapter 6. A fourth and final strategy, namely Best Depth First Search (BeDFS), is to select the most promising node among those at the deepest level of the search tree.

Branching Strategies

We consider the *dichotomic branching* strategy, which consists of selecting the next variable that has not yet been fixed, and fixing it to either 0 or 1.

One important aspect is how to define what is the next variable to be fixed. One option is to consider a fixed order for the variables. In particular, we considered an arbitrary order, as well as the O^{sum}, O^{max}, and O^{min} orders proposed by Bazgan et al. [3]. The latter orders are based on the orders O^j that are induced by the increasing ratios v_i^j/w_i of the items $1 \leq i \leq n$ for each objective $1 \leq j \leq m$. In particular, let r_i^j denote the rank of item i for order O^j. Then, O^{sum} denotes the increasing order induced by the sums of the ranks of each item for orders O^j, that is the increasing order induced by $r_i^1 + \ldots + r_i^m$ for each item i. The order O^{max} gives the increasing order induced by the maximum rank of each item for orders O^j. To break ties, the maximum rank for this order is computed as:

$$r_{\max}(i) = \max_{1 \leq j \leq m} r_i^j + \frac{1}{mn} \sum_{j=1}^{m} r_i^j \tag{2.27}$$

Lastly, O^{min} denotes the increasing order of the minimum rank of each item for orders O^j. To break ties, the minimum rank is computed as:

$$r_{\min}(i) = \min_{1 \leq j \leq m} r_i^j + \frac{1}{mn} \sum_{j=1}^{m} r_i^j \tag{2.28}$$

Our preliminary experiments revealed that the order O^{sum} very often gave better results than the O^{min} and O^{max} orders. As such, we will not show results for these two orders throughout this thesis. Nonetheless, they are implemented along with the algorithms in [33].

Alternatively, it is possible to consider a dynamic order that varies between nodes. However, this is something we did not explore in this thesis. Cerqueus et al. [6] provides a recent overview of other branching orders and strategies for the MOBKP when $m = 2$, some of which can be generalized for more dimensions.

Bound Sets Computation

To compute the lower bound set of a node, i.e., a partial solution to the MOBKP, we first consider the extensions given by Dantzig's greedy algorithm [9] for the single-objective problems given by each objective j, $1 \leq j \leq m$. In particular, assume that the items with index 1 to k have already been fixed for the partial solution of the current node and that the items with index i, $k < i \leq n$, are sorted in non-decreasing order of the ratio v_i^j/w_i for objective j. Then, an

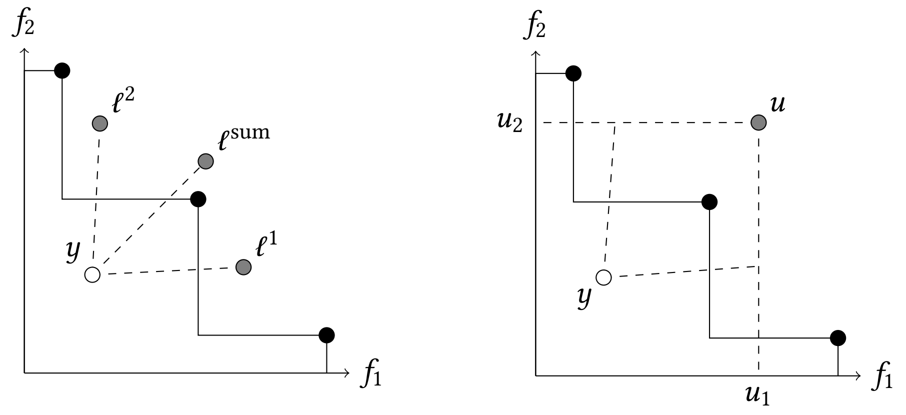


Figure 2.1: Illustration for two objectives of the image of the lower bound set in the objective space $\{\ell^1, \ell^2, \ell^w\}$ and upper bound set $\{u\}$ for a partial solution with objective vector y.

extension for the single-objective problem given by objective j consists of adding all items from $k + 1$ up to, but not including, b such that adding item b would break the weight constraint. The objective vector for such an extension is given by:

$$\ell^j = \left(\sum_{i=1}^{k} x_i v_i^1 + \sum_{i=k+1}^{b} v_i^1, \ldots, \sum_{i=1}^{k} x_i v_i^m + \sum_{i=k+1}^{b} v_i^m \right) \tag{2.29}$$

In addition, to refine the lower bound set with a solution that offers a more balanced trade-off among the objectives, we consider the extension that is found with Dantzig's greedy algorithm for the items with index i, $k < i \leq n$, sorted by the non-decreasing ratio $(v_i^1 + \cdots + v_i^m)/w_i$. The objective vector of this extension is denoted by ℓ^{sum}. We give an illustration of ℓ^1, ℓ^2, and ℓ^{sum} for two objectives on the left-hand side of Figure 2.1.

As for the upper bound set, we consider the same set used by Lukata and Teghem [52]. In particular, for the single-objective problem given by each objective j, $1 \leq j \leq m$, value u_j denotes the upper bound by Martello and Toth [54]. Then, a point u is given by:

$$u = (u_1, \ldots, u_j) \tag{2.30}$$

which characterizes an upper bound set $\{u\}$. This upper bound set is illustrated on the right-hand side of Figure 2.1.

2.3.3 Dynamic Programming

Another class of exact approaches is Dynamic Programming (DP). Several DP approaches have been proposed for the MOBKP, many of which extend the algorithm by Nemhauser and Ullmann [58] for more objectives. Klamroth and Wiecek [42] extended the recurrence formulations of previous DP approaches into a more general framework for the MOBKP. More recently, Bazgan et al. [3] presented a DP approach based on several complementary dominance relations. These dominance relations allow to discard partial solutions that cannot lead to efficient solutions. Figueira et al. [19] proposed some extensions to this algorithm for the bi-objective ($m = 2$) case, by considering a global lower bound set and different techniques to discard partial solutions based on their upper bound. Delort and Spanjaard [12] also considered the use of bound sets to discard partial solutions that cannot lead to efficient solutions for the bi-objective case.

In this work, we consider the Bazgan-Hugot-Vanderpooten Dynamic Programming (BHV-DP) algorithm by Bazgan et al. [3]. This algorithm consists of n stages, such that at any stage k the algorithm generates a set of feasible solutions X_k where the first k items have been set to either 0 or 1, and the remaining items are all set to 0. At each stage, the states are generated according to the following recursion:

$$X_k = \text{Dom}\left(X'_k\right) \tag{2.31}$$

$$X'_k = X_{k-1} \cup \left\{ (\ldots, x_{k-1}, 1, x_{k+1}, \ldots) : \sum_{i=1}^{n} w_i x_i \leq W, x \in X_{k-1} \right\} \tag{2.32}$$

where the initial state $X_0 = \{(0, \ldots, 0)\}$ contains the solution with no chosen items for the knapsack and $\text{Dom}(\cdot)$ denotes one or more dominance relations that can be used to discard solutions that cannot lead to efficient solutions at later stages. The following dominance relations are proposed in [3] for discarding solutions:

- **Dom1:** Let $x, x' \in X'_k$. Then, x can be discarded if x' derives from x at stage k, i.e., if $x' = (\ldots, x_{k-1}, 1, x_{k+1}, \ldots)$, and if $\sum_{i=1}^{n} w_i x_i \leq W - \sum_{i=k}^{n} w_i$.
- **Dom2:** Let $x, x' \in X'_k$. Then, x can be discarded if $f(x') > f(x)$ and $k = n$, or if $f(x') > f(x)$, $k < n$ and $\sum_{i=1}^{n} w_i x'_i \leq \sum_{i=1}^{n} w_i x_i$.
- **Dom3:** Let $x, x' \in X'_k$. Then, x can be discarded if its upper bound set, $\text{UB}(x)$, is strictly dominated by the image of the lower bound set of x', i.e., $\{f(x'') : x'' \in \text{LB}(x')\} \succ \text{UB}(x)$.

Note that, dominance relation **Dom2** is an extension for the multi-objective case of the dominance relation by Nemhauser and Ullmann [58]. For dominance relation **Dom3**, the authors consider a lower bound set with two solutions that are found by Dantzig's greedy algorithm [9] with respect to the orders $O^{\max}$

and O^{sum}. The upper bound set is computed using the upper bound extensions for each objective function by Martello and Toth [54], which has been discussed in the previous section.

For the bi-objective case, we also consider the *B-DP1* extension by Figueira et al. [19], denoted by Figueira-Paquete-Simões-Vanderpooten Dynamic Programming (FPSV-DP). This approach further considers the following dominance relation:

- **Dom4:** Let $x \in X'_k$. Then, x can be discarded if $\mathcal{Y}_{ES} > \mathrm{UB}(x)$, where $\mathcal{Y}_{ES}$ denotes the set of extreme supported solutions that can be computed using the DWS approach.

A possible extension for more dimensions would be to consider the algorithm by Przybylski et al. [66] to compute the $\mathcal{Y}_{ES}$ for more dimensions, or to consider an alternative set of feasible solutions.

2.3.4 Pareto Local Search

Local search approaches keep an archive of solutions that are iteratively explored. At each iteration, a solution that has not yet been explored is selected from the archive, and its neighboring solutions are considered for inclusion in the archive. A neighborhood is defined according to one or more neighborhood mappings of the form $\mathcal{N} : \mathcal{X} \to \mathbb{P}(\mathcal{X})$, where $\mathbb{P}(\mathcal{X})$ is the power set of $\mathcal{X}$, that map a solution to all its neighbors. A local search approach stops once the neighborhood of every solution in the archive has been explored. A Pareto Local Search (PLS) approach [60] considers that the archive contains only mutually non-dominated solutions. Therefore, a neighboring solution is only added to the archive if it is not dominated by any solution currently in the archive. Moreover, when a solution is added to the archive every solution in the archive that becomes dominated by the newly inserted solution is removed.

Algorithm 3 describes a PLS approach for the MOBKP following the *1-flip-exchange* neighborhood described in Liefooghe et al. [48]. This algorithm takes as input a set X_0 of mutually non-dominated solutions. This set can, for example, be generated randomly, generated by a heuristic algorithm, or simply contain a single solution corresponding to the empty knapsack. Sets X_u and X_a denote the set of unexplored mutually non-dominated solutions and the archive of mutually non-dominated solutions, respectively. The `Select` method selects a solution from the set of unexplored solutions X_u. Different selection mechanisms may be considered. For this thesis, we consider that one solution is selected randomly. The *1-flip-exchange* neighborhood considers two neighborhood mappings. The first mapping, denoted by `FlipNeighborhood` in the algorithm, gives all solutions that result from exchanging any item in the current solution. The second mapping, denoted by `ExchangeNeighborhood` in the algorithm, gives all solutions that result from exchanging two items in the current solution. Following the algorithm description in [48], the algorithm only explores the second mapping if no new non-dominated solution was found for the first. It is worth noting that, for this neighborhood exploration definition, the PLS approach may not be able to find the efficient set [48].

Algorithm 3: Pareto Local Search

```
Input: $X_0$ (Set of initial mutually non-dominated solutions)
1  $X_u \leftarrow X_0$
2  $X_a \leftarrow X_0$
3  while $X_u \neq \emptyset$ do
4      $x \leftarrow \texttt{Select}(X_u)$
5      $X_u \leftarrow X_u \setminus \{x\}$
6      aux $\leftarrow 0$
7      for $x' \in \texttt{FlipNeighborhood}(x) \cup \texttt{ExchangeNeighborhood}(x)$ do
8          if $x' \in \texttt{ExchangeNeighborhood}(x) \wedge \text{aux} = 1$ then
9              break
10         end
11         if $\texttt{IsDominated}(x', X_a)$ then
12             continue
13         end
14         $X_u \leftarrow \texttt{RemoveDominated}(X_u, x')$
15         $X_a \leftarrow \texttt{RemoveDominated}(X_a, x')$
16         $X_u \leftarrow X_u \cup \{x'\}$
17         $X_a \leftarrow X_a \cup \{x'\}$
18         if $x' \in \texttt{FlipNeighborhood}(x)$ then
19             aux $\leftarrow 1$
20         end
21     end
22 end
23 return $X_a$
```

2.4 Anytime Algorithms

Anytime algorithms [11, 78], in the context of MOO, can return a set of feasible solutions for any time budget. This means that a DM can trade-off execution time with solution quality. This is relevant in various scenarios, for example, in real-time systems where the available time for the algorithm varies between calls, or in composite systems where the allocated time for each algorithm varies between instances in order to improve the result of the overall system.

A distinction is often made between two categories of anytime algorithms, *interruptible* and *contract* [78]. Interruptible algorithms do not require a priori knowledge of the available time budget and can return a set of feasible solutions when interrupted during their execution. By contrast, contract algorithms require that the time budget is known a priori, and may or may not return a solution if interrupted before that time ends. MOO approaches that keep an archive of feasible solutions, such as the ones described in the previous section, fall into the interruptible category since the archive can always be returned when the algorithm is interrupted. As such, in this thesis we will focus on interruptible algorithms.

To study the performance of a single run of an anytime algorithm we define a *performance trace* that describes the trade-off between solution quality and execution time. In the following, a run refers to a single execution of an algorithm on a particular instance.

Definition 2.10 (Performance trace)**.** The *performance trace* of a run r of an algorithm a is given by a function $Q_{a,r} : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$ that maps execution time to the quality of the solution set that would be returned if the algorithm was interrupted at that time.

A performance trace is said to be monotonic if function $Q_{a,r}(t)$ is non-decreasing for increasing values of t. If every run of an anytime algorithm gives a monotonic performance trace then that algorithm is said to have *monotonic behavior*. Otherwise, the algorithm is said to have *non-monotonic behavior*. Note that algorithms with non-monotonic behavior can typically be made to have monotonic behavior by keeping the archive with the best quality in memory. However, this may not always be possible or desirable. For example, if the objective functions are computationally expensive, the algorithm may use approximate objective functions that are not order-preserving with respect to the true objective functions, which may lead to exchanging better solutions with worse ones. As such, in the following, we want to consider definitions for anytime performance that are meaningful for both monotonic and non-monotonic behaviors.

To characterize the anytime performance of an algorithm for a particular instance, we consider the definition of a *performance profile*.

Definition 2.11 (Performance Profile)**.** The *performance profile* of an algorithm

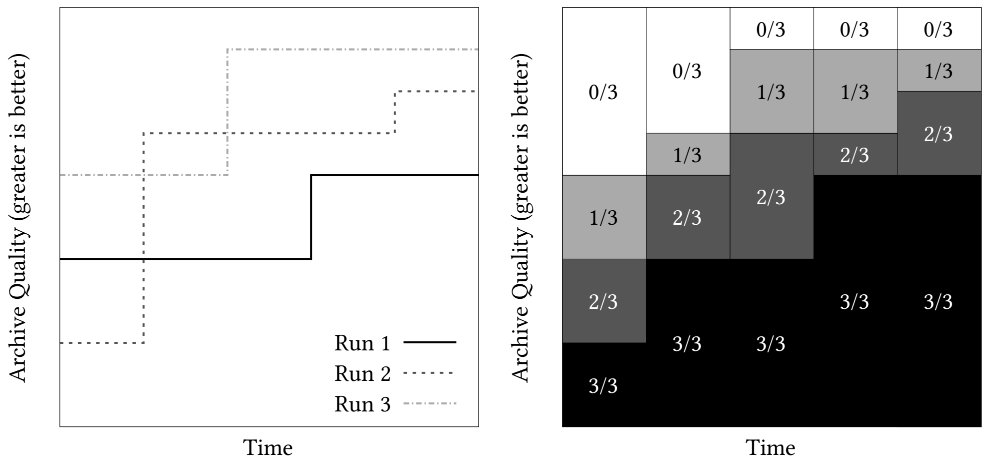


Figure 2.2: Performance traces (left) and profile (right) of three runs of an algorithm.

a on a problem instance ι is given by a function:

$$P_{a,\iota}(t, q) \longrightarrow [0, 1] \tag{2.33}$$

that denotes the probability of a run of algorithm a on problem instance ι finding a solution with quality greater than or equal to q at time t.

We also consider the definition of an *empirical performance profile* which summarizes the anytime performance of an algorithm over a set of runs.

Definition 2.12 (Empirical Performance Profile)**.** For an algorithm a and a set of runs R, an *empirical performance profile* is given by a function:

$$\widehat{P}_{a,R}(t, q) = \frac{1}{|R|} \sum_{r \in R} \begin{cases} 1 & \text{if } Q_{a,r}(t) \geq q \\ 0 & \text{otherwise} \end{cases} \tag{2.34}$$

that denotes the proportion of runs in R that were able to achieve a solution with quality greater than or equal to q at time t.

The definition of similar (empirical) performance profiles as a conditional probability function has been previously considered by Hoos and Stützle [29], denoted by *runtime distribution*, and by Chiarandini [7] through the use of the *empirical attainment function* [25]. Those profiles consider the probability of finding a solution of quality greater than or equal to q at, or before, execution time t. This definition is equivalent to ours if the performance traces of an algorithm are all monotonic, but is different when considering algorithms with non-monotonic performance traces.

Figure 2.2 illustrates the performance traces of three runs of an algorithm on the left, and the corresponding empirical performance profile over those runs on the right. We remark that this gives a similar plot as the empirical attainment function [50].

Lastly, we remark that aggregating performance traces into a single profile should be carefully done [29, 78]. For example, if the considered instances have different quality domains, then their aggregation may not be meaningful. One possibility to mitigate this issue is, for example, to consider a relative measure of quality.

2.5 Algorithm Selection

The ASP [67] concerns the selection of an algorithm to solve a previously unseen instance, such that a given performance measure is optimized. As discussed in Chapter 1, we distinguish between offline algorithm selection, i.e., the problem of selecting an algorithm before attempting to solve the given problem instance, and online algorithm selection, i.e., the problem of selecting and swapping algorithms while solving the given problem instance.

Offline Algorithm Selection

Most existing algorithm selection methodologies fall into the category of offline algorithm selection. Moreover, existing methodologies often consider selecting between single-objective optimization algorithms rather than MOO algorithms, and do not consider the anytime preferences of the DM at the time selection. We refer to Kotthoff [44] and Kerschke et al. [41] for two recent reviews of existing offline algorithm selection methodologies.

These methodologies are generally divided into two categories, *regression* and *classification*. Regression approaches predict the performance measure for each algorithm and select the algorithm with maximal value, see for example, Leyton-Brown et al. [46] and Xu et al. [77]. Classification approaches instead select an algorithm without predicting the performance measure, for example, by learning which algorithm gives the best performance measure for different kinds of instances and utility functions, see for example, Vilas Boas et al. [74] and Polyakovskiy et al. [64].

Note that, most offline algorithm selection methodologies make use of *instance features* to characterize a problem instance as a set of measurable values. For single-objective optimization problems many, general-purpose and problem-specific, instances features have been proposed, e.g., Nudelman et al. [59] and Mersmann et al. [55, 56]. For MOO, there has been less research. However, there are some recent promising developments with respect to general-purpose instance features, e.g., Daolio et al. [10] and Liefooghe et al. [47, 49].

Online Algorithm Selection

Online algorithm selection methodologies are less common than offline, but existing methodologies similarly focus on selecting between single-objective optimization algorithms. Some methodologies monitor the performance of the algorithms to detect which ones are more promising and should be given more time to run, e.g., Gagliolo et al. [23] and Gagliolo and Schmidhuber [22]. Other approaches, consider selecting between algorithms for exploring a search tree, e.g, Arbelaez et al. [2] selects the best search strategy at checkpoints in the search tree. Up to our knowledge, no online algorithm selection methodology has focused on selecting between (anytime) MOO algorithms. We refer to Kotthoff [44] for a review of existing online algorithm selection methodologies.

Related Topics

Lastly, two related areas of research are meta-level control and composition of anytime algorithms [78]. Meta-level control, in the context of anytime algorithms, relates to the problem of deciding when to interrupt an anytime algorithm and act on the returned solution [28, 71]. Composition of anytime algorithms, denotes the problem of building a complex system comprised of one or more anytime algorithm that maximizes the overall performance of the system [68]. Up to our knowledge, these problems have not been considered in the context of MOO problems.

2.6 Conclusion

In this chapter, we presented the theoretical background for multi-objective optimization, and discussed several algorithms for the MOBKP, a benchmark problem that will be used throughout this thesis.

We also discussed the concepts related to anytime algorithms and anytime performance since most MOO algorithms naturally fall into this category of algorithms. Another important aspect to note is that, for non-trivial problem instances, MOO exact approaches cannot commonly find the efficient set in a feasible amount of time, and even heuristic algorithms may take too long to complete in some cases. Moreover, preferences with respect to time budget and target quality may not be known when developing the algorithms or training the model, but rather only when selecting the best algorithm to solve a particular problem instance. As such, it is often more relevant to look at MOO algorithms as anytime algorithms, and consequently to model and study their anytime performance rather than simply looking at their performance at the end of execution or at a fixed time budget or target quality. This motivates us to look at the anytime performance of the MOO algorithms when tackling the ASP.

Lastly, we provided some context regarding the ASP. In particular, we highlighted that the ASP has been studied for the selection of single-objective algorithms, often considering fixed time budgets or target qualities, over several decades. However, researchers have only recently started looking at the ASP in the context of MOO algorithms. Moreover, these recent works have only considered offline selection with time budgets that were set before training the selection model. Still, these recent developments, and the positive results that are known for the selection of single-objective algorithms, show that this is a relevant problem to address.

Chapter 3

Theoretical Model of Anytime Performance

Theoretical models can be used to predict the performance of an algorithm on previously unseen instances without the need to empirically evaluate the algorithm on a set of training instances beforehand. In this chapter, we give a theoretical model to approximate the ideal anytime performance of bi-objective optimization algorithms that (attempt to) collect an efficient solution at each iteration [38, 39]. Examples are the scalarization techniques introduced in Section 2.3.1. Anytime performance is considered with respect to the number of iterations and the quality of the archive of collected solutions measured with the hypervolume. We also propose an ε-constraint approach guided by this theoretical model [38] to define the scalarized subproblems that are to be solved at each iteration. The experimental results show that the theoretical model can be used to predict the ideal anytime performance, as well as the anytime performance of the proposed ε-constraint approach.

This chapter is organized as follows. In Section 3.1, we present the theoretical model, and carry out an experimental study to analyze its quality against the ideal anytime performance. In Section 3.2, we give the ε-constraint approach guided by the theoretical model, and carry out an experimental study to analyze its anytime performance. In Section 3.3, we conclude with a summary of the main findings in this chapter and discuss possible directions for future research.

3.1 Theoretical Model of Anytime Performance

Consider an iterated algorithm to solve a MOO problem that finds an (efficient) solution to the problem at each iteration, and keeps an archive of all mutually non-dominated solutions found. We describe this algorithm by a function A that returns the set of solutions found after $i > 0$ iterations:

$$A(i) = A(i-1) \cup x^i \tag{3.1}$$

where $A(0) = \emptyset$, and $x^i \in \mathcal{X}$ denotes the efficient solution found at iteration i. The performance trace of this algorithm with respect to the number of iterations and the hypervolume of the image of the archive in the objective space, is given by:

$$Q(i) = I_H(\{f(x) : x \in A(i)\}) \tag{3.2}$$

where $f(x)$ denotes the multi-objective function being optimized.

Under the assumption that the number of iterations of the algorithm is unknown, we define an *ideal algorithm* A^* with complete knowledge of the efficient set $\mathcal{X}_E$ that finds, at each iteration, the solution that contributes the most to the hypervolume of the archive, that is:

$$A^*(i) = A^*(i-1) \cup \underset{x \in \mathcal{X}_E}{\operatorname{argmax}}\, I_H\left(\{f(x') : x' \in A^*(i-1) \cup \{x\}\}\right) \tag{3.3}$$

We assume, for the sake of simplicity, that there exists a single optimum for the problem in the right-hand side of the equation above. Nonetheless, if there are multiple optima, the formulation can be extended to consider the list of all sets that maximize the hypervolume at each iteration. The performance traces of this ideal algorithm gives the *ideal performance trace*, which can be described by the following function:

$$Q^*(i) = I_H(\{f(x) : x \in A^*(i)\}) \tag{3.4}$$

Note that this definition of an ideal anytime performance trace does not necessarily give the maximal hypervolume that can be achieved with i efficient solutions. However, a definition that gives, for each iteration, the maximal hypervolume is often incompatible with an algorithm that collects, at each iteration, a single efficient solution, since the sets of solutions for two consecutive iterations can differ by more than one solution.

Finding the ideal performance trace requires knowing the efficient set, which is unrealistic when solving a previously unseen problem instance. Instead, we define a theoretical model to approximate the ideal performance trace for bi-objective optimization problems that does not require such knowledge. This model works in two phases. In the first phase, we define a piecewise linear approximation to the non-dominated set under some assumptions. In particular, we assume that the objective values of the lexicographic optimal solutions are known and that the non-dominated set can be well approximated by the positive quadrant of a superellipse. Although it is not expected that the non-dominated set matches the positive quadrant of a superellipse exactly, our findings suggest that this gives a good approximation in practice for many problems with linear sum objective functions. Emmerich and Deutz [18] have also considered the use of superellipses for the generation of MOO test problems.

In the second phase, we give two methodologies to compute the performance trace of an *oracle* that collects, at each iteration, a point in the piecewise linear approximation that maximizes the hypervolume. This oracle is characterized by the increase in hypervolume at each iteration, denoted by $C(i) \to \mathbb{R}$, which corresponds to the hypervolume contribution of the newly found point to the archive. As a result, the performance trace for this oracle, which approximates the ideal performance trace, is given by:

$$\widetilde{Q^*}(i) = \sum_{i=1}^{n} C(i) \tag{3.5}$$

The first methodology that we give to compute the values of $C(i)$, is an analytical formulation that requires several assumptions to hold. In particular, we assume that the piecewise linear approximation is convex and that it is given by two symmetric linear segments. Moreover, we assume that the reference point for the hypervolume indicator is given by the *nadir point* of the non-dominated set, i.e., the point given by the minimum coordinate values for points in the non-dominated set. The second methodology is an algorithm to compute $C(i)$ for any piecewise linear approximation, and for any reference point setting. Despite being more general, this algorithmic approach is computationally expensive. However, empirical results show that the algorithmic methodology is still very fast in practice, so this is unlikely to be a problem for most applications.

The remainder of this section is organized as follows. In Section 3.1.1, we define the piecewise linear approximation to the non-dominated set. In Sections 3.1.2 and 3.1.3, we present the analytical and algorithmic methodologies respectively. In Section 3.1.4, we present an empirical study comparing the performance trace given by our theoretical model against the ideal performance trace.

3.1.1 Estimating the non-dominated set

Let us assume that the non-dominated set for a bi-objective optimization problem with maximizing objectives, scaled down to the unit square $[0, 1]^2$, can be well approximated by the positive quadrant of a superellipse centered in the origin with both semi-diameters of length one. Formally, this is given by the

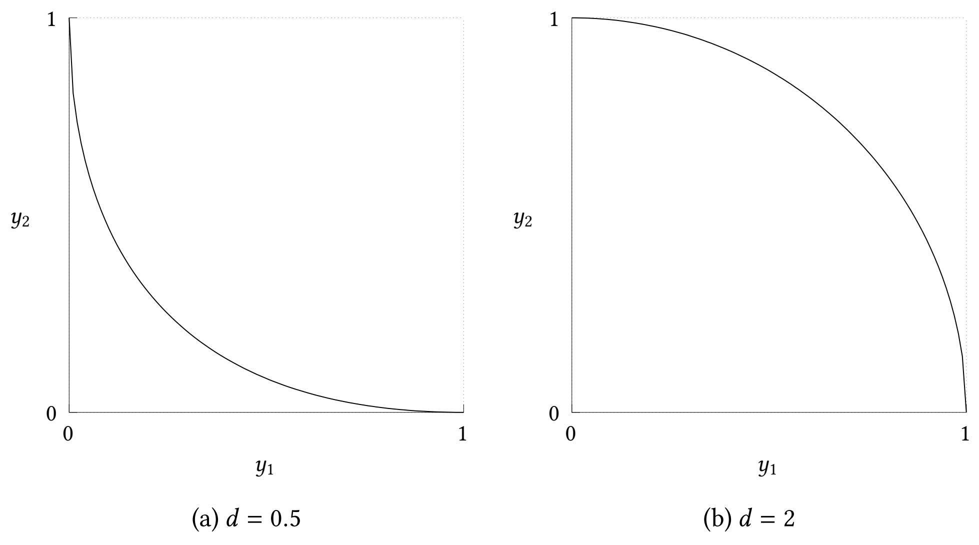


Figure 3.1: Example of the superellipse curve for two distinct values of the curvature parameter $d \in \{0.5, 2\}$.

following parametric equation:

$$y_1{}^d + y_2{}^d = 1 \tag{3.6}$$

where $d > 0$ is a parameter that controls the curvature of the superellipse and $y_1, y_2 \in [0, 1]$. Alternatively, the values of y_1 and y_2 can also be defined with respect to a parameter $\theta \in [0, \pi/2]$ as follows

$$y_1 = \cos^{2/d}\theta \tag{3.7}$$

$$y_2 = \sin^{2/d}\theta \tag{3.8}$$

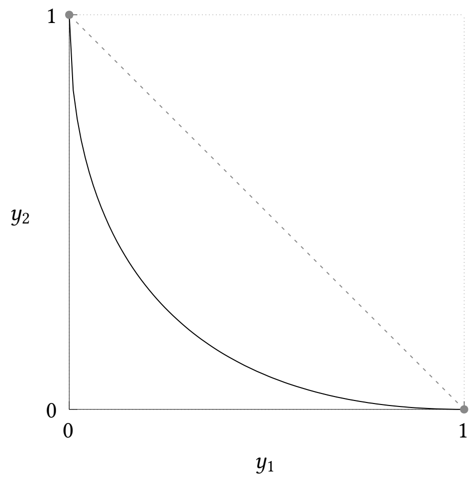
Note that, for $d < 1$ the resulting approximation is non-convex, whereas for $d \geq 1$ it is convex. This is illustrated in Figure 3.1 where we show the approximation for two distinct values of d.

For our theoretical model, we consider an approximation to the positive quadrant of the superellipse defined by a piecewise linear approximation G with $\ell > 0$ linear segments defined between consecutive points in the curve approximation, such that each point g_i, $i \in \{1, \ldots, \ell + 1\}$ can be defined by:

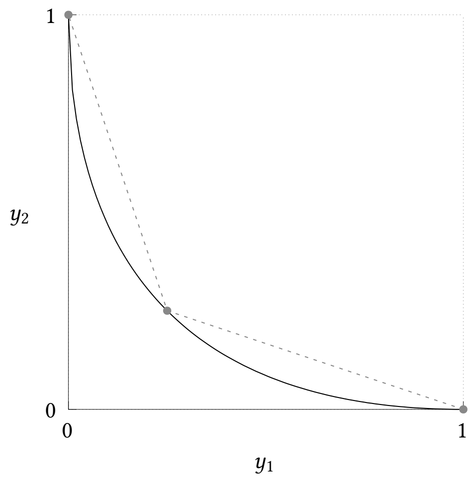
$$g_i = \left(\cos^{2/d}\theta_i, \sin^{2/d}\theta_i\right) \tag{3.9}$$

where $\theta_i < \theta_{i+1}$ and the values of θ_i are evenly spaced in the interval $[0, \pi/2]$. We found that this simple setting of θ_i provides a good approximation for the purposes of our model. However, other techniques are available to sample these points, e.g., see [62].

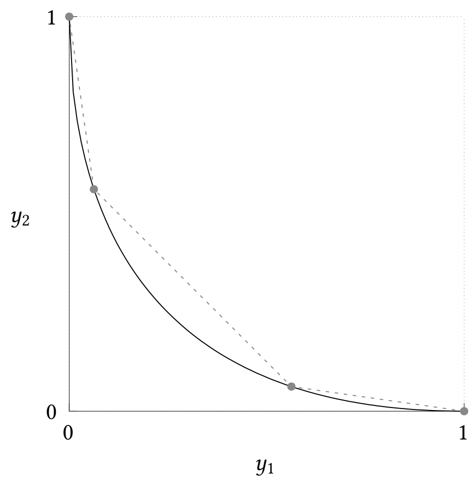
In Figures 3.2 and 3.3, we plot the superellipse curve given by Equation (3.6) for two distinct curvature parameters $d \in \{0.5, 2\}$, as well as the corresponding piecewise linear approximations G for a varying number of segments ℓ.



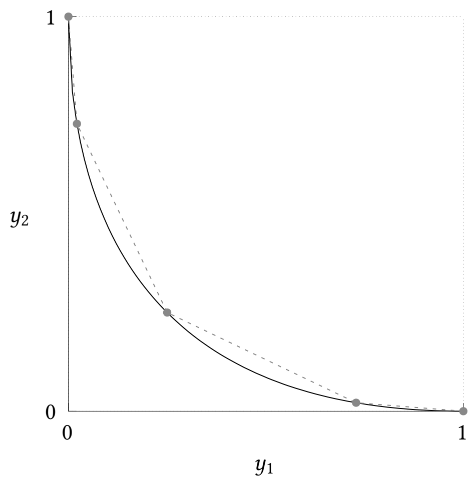
(a) $\ell = 1$



(b) $\ell = 2$



(c) $\ell = 3$



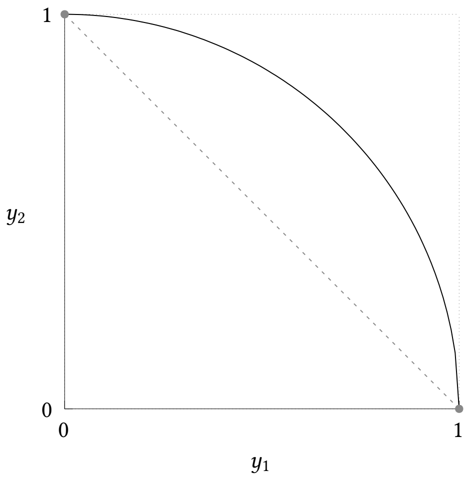
(d) $\ell = 4$

Figure 3.2: Example of the superellipse curve (continuous line) for $d = 0.5$ and the corresponding piecewise linear approximation (dashed line) for a varying number of segments $\ell \in \{1, 2, 3, 4\}$.

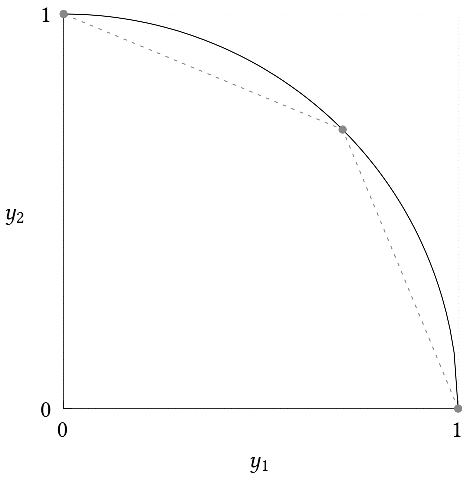
As expected, the piecewise linear approximation better approximates the superellipse curve as the number of segments increases. Moreover, for $d < 1$ the piecewise approximation weakly dominates the superellipse curve and will therefore overestimate the area dominated by the curve, whereas for $d > 1$ it will underestimate it.

3.1.2 Analytical methodology

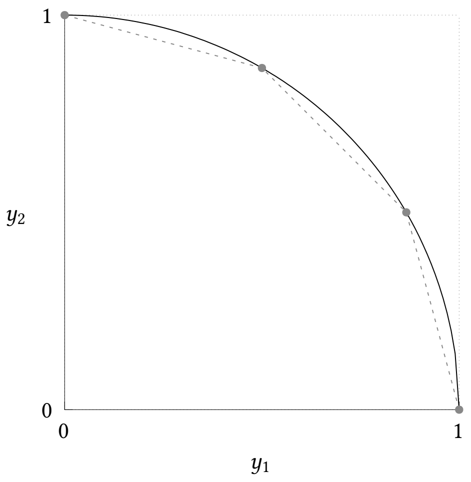
Under some assumptions, we can define a closed formula for the performance trace of an oracle that collects, at each iteration, a point in the piecewise linear approximation G that maximizes the hypervolume. In particular, we assume



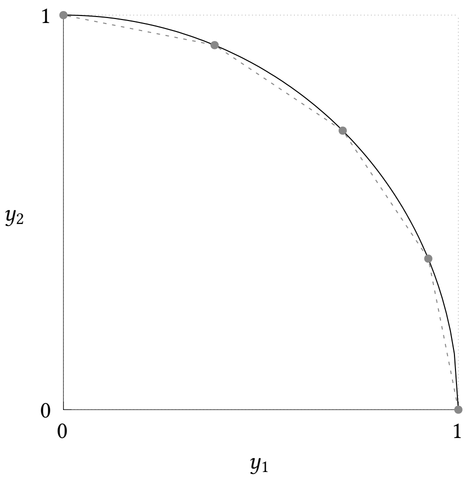
(a) $\ell = 1$



(b) $\ell = 2$



(c) $\ell = 3$



(d) $\ell = 4$

Figure 3.3: Example of the superellipse curve (continuous line) for $d = 2$ and the corresponding piecewise linear approximation (dashed line) for a varying number of segments $\ell \in \{1, 2, 3, 4\}$.

that the piecewise linear approximation is convex, i.e., $d \geq 1$, and that it has two linear segments such that $\theta_1 = 0$, $\theta_2 = \pi/4$, and $\theta_3 = \pi/2$. Moreover, the reference point for the hypervolume indicator is given by the *nadir point* of the piecewise linear approximation, i.e., $r = (0, 0)$. Under these assumptions, the piecewise linear approximation can be defined as:

$$G = \{(y_1, g(y_1)) \mid y_1 \in [0, 1]\} \tag{3.10}$$

$$g(y_1) = \begin{cases} \frac{p-1}{p} y_1 + 1 & , 0 \leq y_1 \leq p \\ \frac{p}{p-1} y_1 + \frac{p}{1-p} & , p < y_1 \leq 1 \end{cases} \tag{3.11}$$

where

$$p = \cos^{2/d} \frac{\pi}{4} = \sin^{2/d} \frac{\pi}{4} \tag{3.12}$$

The first point collected by the oracle for such a piecewise linear approximation is found by solving the problem:

$$\underset{y_1 \in [0,1]}{\operatorname{argmax}} \; y_1 \cdot g(y_1) \tag{3.13}$$

This problem has a single global optimum at coordinates (p, p). As such, the first point collected by the oracle gives a contribution $C(1) = p^2$.

Subsequent points can be found by considering the regions that are not dominated by any of the points previously collected. These regions are said to be *uncovered*. Regions dominated by at least one of the previously collected points are said to be *dominated*. After the first point (p, p) has been collected, there are two uncovered regions each defined by a right triangle such that the hypotenuse corresponds to the non-dominated set of that uncovered region. In Figure 3.4, we illustrate the uncovered and dominated regions, colored in gray and black respectively, after collecting one point (left-hand side) and three points (right-hand side).

To find the largest hypervolume given by the dominated area of a point in an uncovered region defined by a right triangle such that the set of non-dominated points is given by the hypotenuse and the reference point is on the right angle, we solve the following problem:

$$\begin{aligned} \max \;\; & y_1 \cdot y_2 \\ \text{s.t.} \;\; & y_2 = -\frac{c_2}{c_1} \cdot y_1 + c_2 \\ & y_1 \in [0, c_1] \\ & y_2 \in [0, c_2] \end{aligned} \tag{3.14}$$

where c_1 and c_2 denote the length of the catheti of the right triangle on the y_1 and y_2 axes, respectively. This problem has a global optimum of $c_1 \cdot c_2/4$

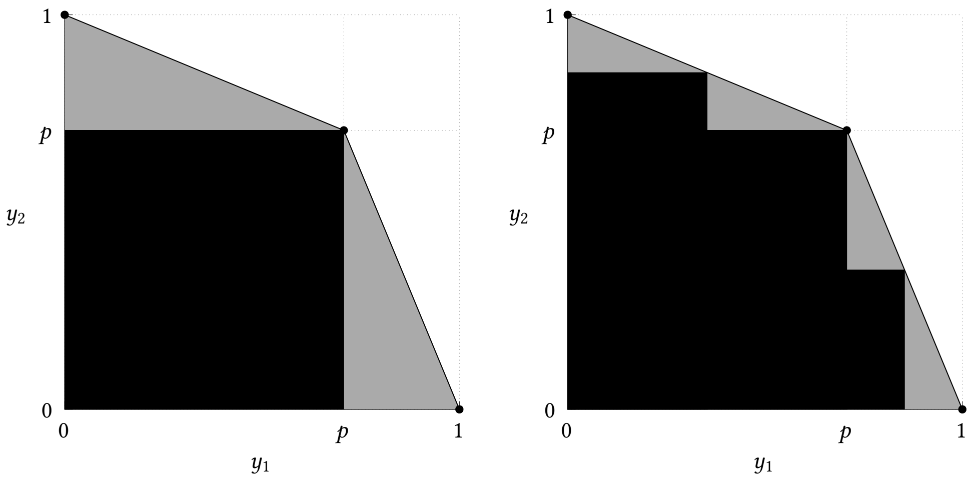


Figure 3.4: Example of the dominated (in black) and uncovered (in gray) regions after collecting one (left) and three (right) points with the oracle.

at point ($y_1 = c_1/2, y_2 = c_2/2$). Note that, the hypervolume given by the dominated area of this point corresponds to half the area of the uncovered region. Moreover, after collecting this point and considering its dominated area, there are two smaller uncovered regions that are also right triangles, each of which with catheti that are half the length of those in the original triangle, and consequently covering a quarter of the area.

After collecting the first point, the resulting uncovered regions are two equivalent right triangles with catheti of size $(1 - p)$ and p, and an area of $(1 - p) \cdot p/2$. Then, from Equation (3.14), it follows that the second and third contributions for points collected by our oracle are $C(2) = C(3) = (1-p) \cdot p/4$. After excluding the regions dominated by the two points providing these contributions, one for each uncovered region, there are four equivalent uncovered regions as illustrated on the right-hand side of Figure 3.4. After collecting the points that maximize the hypervolume in all equivalent uncovered regions, the number of uncovered regions doubles. As such, a general equation for $C(i)$ is given by

$$C(i) = \begin{cases} p^2 & i = 1 \\ \frac{(1-p)\cdot p}{4^{\lfloor \log_2 i \rfloor}} & i > 1 \end{cases} \tag{3.15}$$

which shows a logarithmic rate of convergence. Computing the performance trace of the oracle, given by $\widetilde{Q}^*(j)$ until iteration $j > 0$, which is the result of the sum of $C(i)$, $0 < i \leq j$, can be done in $O(j)$ if the value of $\lfloor \log_2 \ i \rfloor$ is kept updated in constant time.

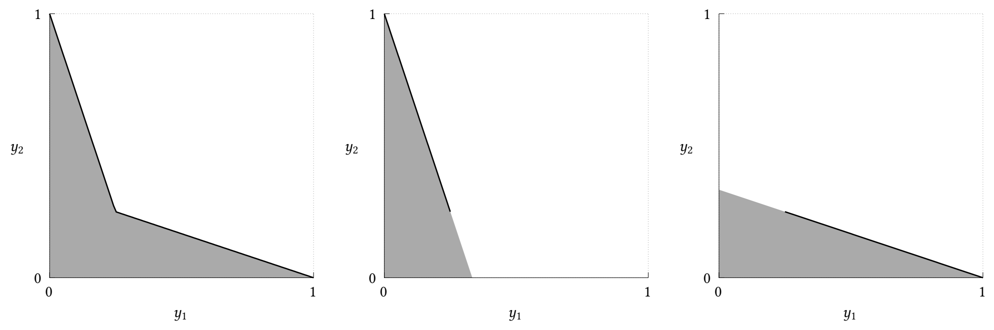


Figure 3.5: On the left, an uncovered region defined by a simple polygon (gray area). On the middle and on the right, the individual segments (black line) of the polygonal chain, and their respective right triangles (gray area).

3.1.3 Algorithmic methodology

For the general case, i.e., for any number of linear segments $\ell > 0$, any curvature parameter $d > 0$, and any reference point $r \in \mathbb{R}^2$, the previous analytical model does not hold. Instead, we present an algorithm to compute the contributions $C(i)$, for iterations $i > 0$, which works by keeping an updated set of the uncovered regions. Note that, only one point from a single uncovered region is collected at each iteration, and that the uncovered regions share no area. As such, to avoid repeated calculations, the algorithm keeps a cache of the points in each uncovered region that contribute the most to the hypervolume of the archive. Moreover, to efficiently find the largest contribution, the uncovered regions can be kept in a priority queue with respect to the largest contribution given by a point in that region.

For the purposes of our algorithm, an uncovered region that results from the piecewise linear approximation can be described by a simple polygon with a right angle on the point with the smallest coordinates, i.e., the reference point for that uncovered region, such that the non-dominated set is given by the simple polygonal chain opposite to that point. As a result, each segment of the polygonal chain is part of the hypotenuse of a right triangle with its right angle on the reference point. In Figure 3.5 we give an illustration of an uncovered region with a polygonal chain made up of two segments, and the corresponding right triangles for each segment.

We know from Equation (3.14) that the point with the largest hypervolume for a right triangle with the reference point on the right angle is given by $z = (c_1/2, c_2/2)$, where c_1, c_2 are the lengths of the catheti on axes y_1, y_2, respectively. Moreover, the result for that optimization problem is strictly monotonically increasing for $0 \le y_1 \le c_1/2$, and strictly monotonically decreasing for $c_1/2 \le y_1 \le c_1$. As such, the point in a segment of the polygon chain with

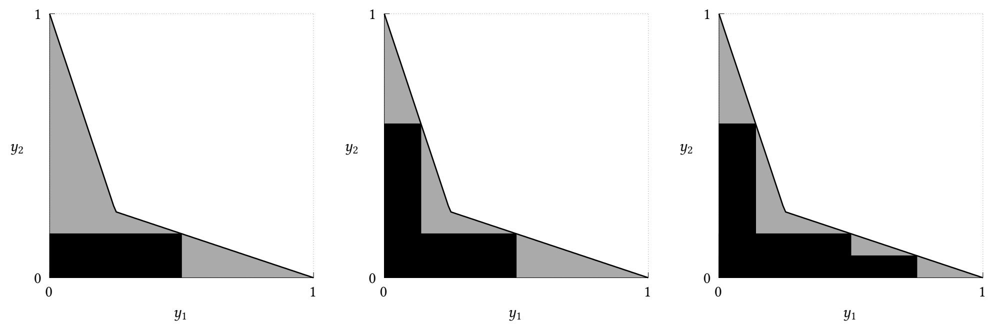


Figure 3.6: First three steps of the algorithmic methodology for a curvature parameter $d = 0.5$, a number of linear segments $\ell = 2$, and a reference point $r = (0, 0)$.

the largest hypervolume is the point in that segment closest to z. Finally, we can iterate over all segments of the polygonal chain to find the point with the largest hypervolume for the whole uncovered region. In Figure 3.6, we show the first three steps of the algorithm for a non-convex piecewise approximation with $\ell = 2$ linear segments.

Note that there may be multiple points that give the same hypervolume at each iteration. This is the case for the first step illustrated in Figure 3.6 where the points $(0.5, 1/6)$ and $(1/6, 0.5)$ both give the same hypervolume for a reference point $(0, 0)$. If there are only two points, and the new uncovered regions that result from selecting either point are equivalent, then we can select either point since the progression of $C(i)$ will be the same regardless of the choice. We say that two uncovered regions are equivalent, if they have the same, possibly mirrored, shape. This is the case for the first step illustrated in Figure 3.6, where selecting the first point leads to uncovered regions that are mirrored by those that result from selecting the second point. By contrast, if the resulting uncovered regions are not equivalent, then we need to consider both choices until a point is found at a later iteration that results in a greater contribution $C(i)$ for either of those choices. We can do this by keeping a list of priority queues to be processed at each iteration, such that each priority queue denotes an alternative choice. The number of priority queues in this list may grow exponentially since we have at most two choices at each iteration. However, in our experimental analysis we never observed more than one or two priority queues in the list at any given time, which suggests that an exponential scenario is unlikely to occur.

If more than two points give the same contribution, then we show that we only need to consider choosing between the two lexicographically extreme points. Let $Z \subseteq G$ be a list of at least three distinct lexicographically sorted points that give the same contribution $C(i)$ for an iteration $i > 0$. Let $Z^{\min}$ and $Z^{\max}$ be the lexicographically smallest and largest points respectively. Since the points in Z are mutually non-dominated and lexicographically sorted, then $Z^{\min}$ has the smallest first coordinate, and $Z^{\max}$ the smallest second coordinate. Also, let $r \in \mathbb{R}^2$ be the reference point for the hypervolume such that it is strictly dominated by every point in the polygonal chain. Then, for any point y in the polygonal chain G that can be selected in the succeeding iteration and has a larger first coordinate than the previously selected point, it holds that:

$$\forall z \in Z \setminus \{Z^{\min}\}\ y_1 > z_1 \implies (y_1 - Z_1^{\min})(y_2 - r_2) > (y_1 - z_1)(y_2 - r_2) \quad (3.16)$$

since $Z_1^{\min} < z_1$. This implies that, if the succeeding point has a larger first coordinate than the previously selected point, then choosing $Z^{\min}$ in the previous iteration would yield a strictly larger contribution for y. Likewise, if point y has a smaller first coordinate than the previously selected point, it holds that:

$$\forall z \in Z \setminus \{Z^{\max}\}\ y_1 < z_1 \implies (y_1 - r_1)(y_2 - Z_2^{\max}) > (y_1 - r_1)(y_2 - z_2) \quad (3.17)$$

since $Z_2^{\max} < z_2$. This implies, that if the succeeding point has a smaller first coordinate than the previously selected point, then choosing $Z^{\max}$ in the previous iteration would yield a strictly larger contribution for y. As we have covered every possible succeeding point $y \in G$, and either of the two lexicographically extreme points always give a strictly better hypervolume for y, then we can discard the other points.

To illustrate this, consider Figure 3.7 where we show the uncovered regions after selecting one of the three points $Z^1 = (0.08, 0.5), Z^2 = (0.2, 0.2), Z^3 = (0.5, 0.08)$ that give the same hypervolume for a reference point $(0, 0)$. We observe that any subsequent point returned in the uncovered regions A and B resulting from choosing point Z_2, will have a strictly larger contribution in the uncovered regions A' and B' that result from choosing the lexicographically largest or smallest points Z_3 or Z_1 respectively.

In the following we give the complexity of our algorithm for computing $C(i)$. At each iteration, the algorithm needs to find an uncovered region with a point that gives the largest contribution, remove it from the priority queue containing all uncovered regions, split the region into two new uncovered regions and insert those regions into the data structure. As such, for the i-th call, there are at most i uncovered regions in the data structure that need to be considered. Since the uncovered regions share no area, we only need to compute the points with the largest contribution in a given region once. Then, we can keep the uncovered regions in a priority queue supported by max-heap data structure. Removing the uncovered region with largest hypervolume

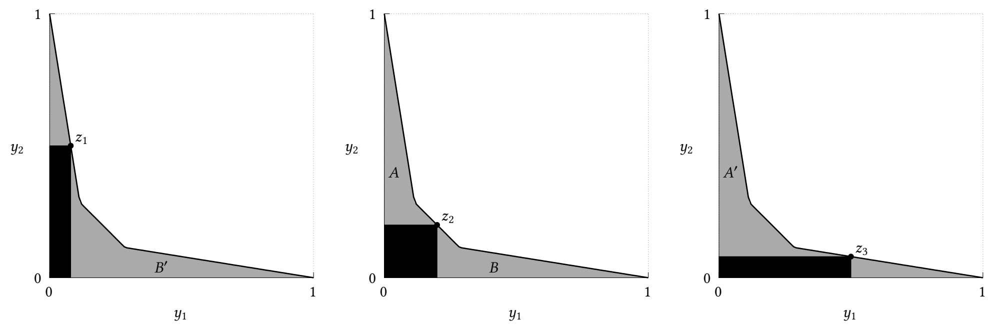


Figure 3.7: Uncovered regions that result from choosing one of three points with the same hypervolume.

contribution from such a data structure can be done in $O(\log i)$ time, and insert new uncovered regions in $O(1)$ or $O(\log i)$ time depending on the max-heap data structure chosen.

To find the points in each uncovered region that give the largest hypervolume we need to loop over all segments and find the point for each segment with maximal hypervolume. This can be done in constant time for each segment, as discussed previously, and there are at most ℓ segments in each uncovered region, so this takes $O(\ell)$ time for each region.

Finally, there may be at most 2^i priority queues to be processed at each iteration if multiple points yield the same contribution. As a result our algorithm has a worst-case time complexity given by $O(2^i(i\ell + i\log i))$ for function $C(i)$. To approximate $P^*(j)$ we can compute $C(j)$, and sum all the intermediate values $C(i), 1 \leq i \leq j$, so the worst-case time complexity is also given by $O(2^j(j\ell + j\log j))$. Alternatively, we consider a greedy variant that always selects the lexicographically smallest point when multiple points have the same hypervolume. The worst-case time complexity of this greedy variant is given by $O(j\ell + j\log j)$.

3.1.4 Empirical study

A `C++` implementation of the exact and greedy algorithmic methodologies is available at [31]. In Figure 3.8, we report the average execution times of this implementation for a varying number of iterations, a varying number of linear segments, and curvature parameter $d \in \{0.5, 2\}$. The reference point is set to $r = (0, 0)$ since it had a negligible impact on the results. The execution times were taken with the `benchmark` library [4] (v.1.6.0) considering a `MinTime` of 10 seconds, which broadly means that for each combination of parameters the methodologies were repeatedly executed until CPU time was greater than 10 seconds. The experiments were carried out in a computer with an AMD

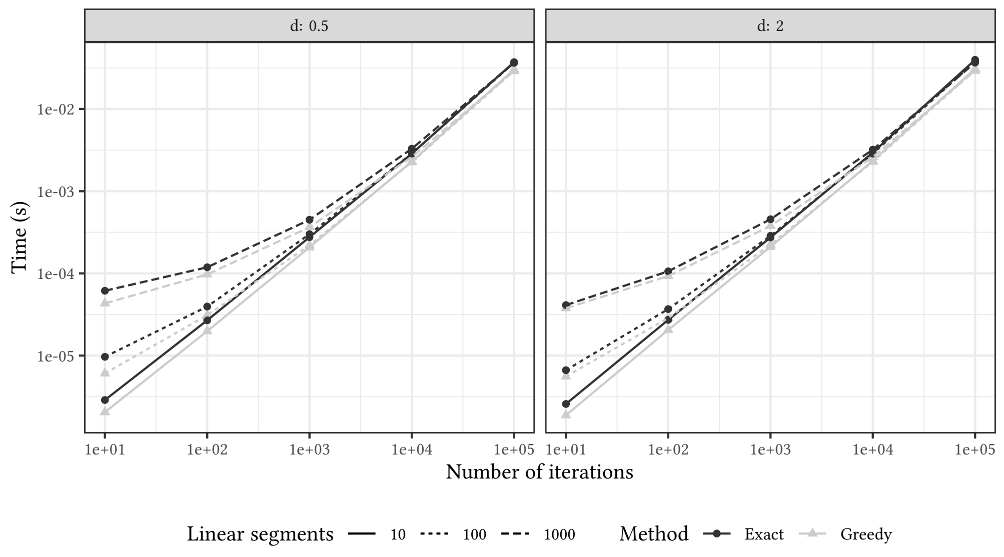


Figure 3.8: Execution times in seconds of the algorithmic methodology for a varying number of iterations, varying number of linear segments, reference point $r = (0, 0)$, and curvature parameter $d \in \{0.5, 2\}$.

Ryzen CPU 5 1600 3.2Ghz processor. The benchmark code and full results are available at [32].

The results show that both variants are very fast (less than 0.1 seconds on average) for the given parameters. We expect most practical scenarios to have a number of linear segments and iterations within the values reported. There is a small difference between the exact and greedy methodologies since the exact methodology has to take into account the possibility of more than one point giving the same maximal contribution. Moreover, if there are multiple points, which in fact happens for the first iteration of $d = 0.5$, the exact methodology attempts to detect if the uncovered regions left after selecting each of those points are equivalent. In this case, the uncovered regions are equivalent. As such, only a single point needs to be considered and the impact is relatively small. Otherwise, the impact on performance could be greater. However, as previously mentioned, in our experiments we did not observe such cases.

In the following, we analyze the quality of our theoretical model on non-dominated sets that arise from particular MOO problems. First, we consider non-dominated sets arising from the Unconstrained Binary Knapsack Problem (UBKP) scaled down to the unit square $[0, 1]^2$, which is formally given by:

$$\underset{x \in \{0,1\}^n}{\text{argmax}}\ f(x) = \left(f_1(x) = \frac{\sum_{i=1}^n v_i x_i}{\sum_{i=1}^n v_i},\ f_2(x) = 1 - \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i} \right) \tag{3.18}$$

where x_i is a binary variable denoting whether or not item i has been chosen for the knapsack, $v_i \in [0, 1]$ denotes a value for that item, and $w_i \in [0, 1]$ denotes a weight for the item. The goal, is to maximize the sum of the values and minimize the sum of the weights of the chosen items. The values and weights of the items are randomly generated according to a multivariate uniform distribution in the range $[0, 1]$, with correlation $\rho \in [-1, 1]$ following the procedure described in [72]. The non-dominated set can be found by the DP algorithm of Nemhauser and Ullmann [58].

Figures 3.9a to 3.9c shows the non-dominated set for instances with correlation $\rho \in \{-0.8, 0.0, 0.8\}$ and problem size $n = 100$. We see that for greater values of ρ the shape of the non-dominated set almost resembles a line, and that for smaller values of ρ the non-dominated set resembles a more accentuated convex curve. The same figures show that the non-dominated set can be well approximated with the positive quadrant of a superellipse with curvature parameter $d = \log_p 0.5$, where p is found by considering the solution to the following optimization problem:

$$\underset{x\in\{0,1\}^n}{\operatorname{argmax}} \ \min\{f_1(x), f_2(x)\} \tag{3.19}$$

such that if x' is the optimal solution to this formulation, then the parameter is given by $p = (f_1(x') + f_2(x'))/2$.

We remark that the non-dominated sets generated for the UBKP are always approximated by a convex curve. As such, to validate our model for non-convex curves, we consider a variant of this problem characterized by a tight capacity constraint on the number of items to be included in knapsack. We refer to this variant as the Capacity Constrained UBKP (CCUBKP), which, scaled down to the unit square $[0, 1]^2$, is formally defined as:

$$\underset{x\in\{0,1\}^n}{\operatorname{argmax}} f(x) = \left(f_1(x) = \frac{\sum_{i=1}^n v_i x_i}{\max_{j\in\{1,\ldots,n\}} v_j},\ f_2(x) = 1 - \frac{\sum_{i=1}^n w_i x_i}{\max_{j\in\{1,\ldots,n\}} w_j} \right) \tag{3.20}$$

$$\text{s.t.} \ \sum_{i=1}^{n} x_i \leq 1 \tag{3.21}$$

This problem is trivial to solve since the set of feasible solutions contains the solutions that result from adding a single item to the knapsack, or none. However, it allows us to devise instances such that the non-dominated set is non-convex. In particular, instances for this problem are generated by sampling the value vector v and an auxiliary weight vector w' from equally parameterized Weibull distributions for a fixed scale parameter $\lambda = 1$, and a varying shape parameter s. Then, the value vector is sorted in increasing order and the auxiliary weight vector is sorted in decreasing order. The weight vector values are then given by $w_i = (\max_{j\in\{1,\ldots,n\}} w'_j) + (\min_{j\in\{1,\ldots,n\}} w'_j) - w'_i$, $i \in \{1, \ldots, n\}$. The non-dominated set for the CCUBKP is trivially found by enumerating all objective vectors.

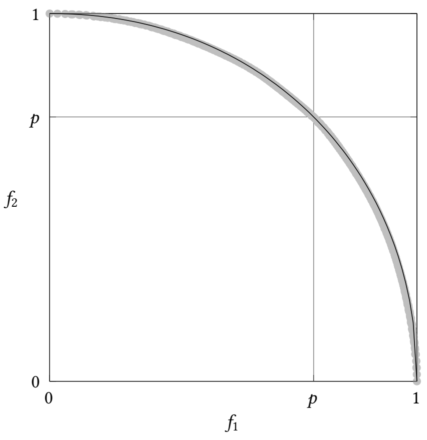
Note that, by varying the shape parameter s, we vary the curvature of the non-dominated set. In particular, for smaller values of s the curvature is more accentuated, whereas for larger values of s the curvature is less accentuated. In Figures 3.9d to 3.9f we plot the non-dominated sets that arise in problem instances with shape parameter $s \in \{1.6, 2.0, 3.0\}$ and problem size $n = 2000$. A larger problem size for the CCUBKP was chosen in order to have a similar number of non-dominated points between the two problems. The plots show that when s increases the non-dominated sets generated for the CCUBKP can be well approximated with the positive quadrant of a superellipse. The curvature parameter for the superellipse approximation is given by $d = \log_p 0.5$, where p is found by considering the solution to the following optimization problem:

$$\underset{x \in \{0,1\}^n}{\text{argmax}} \ \min \{f_1(x), f_2(x)\} \tag{3.22}$$

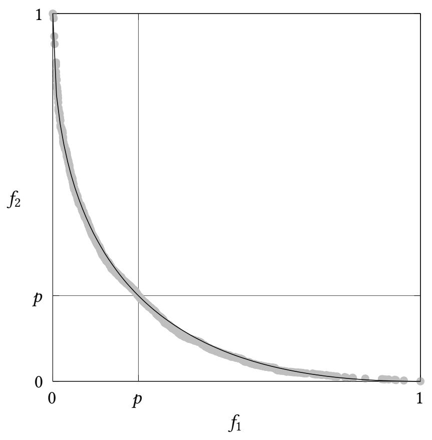
$$\text{s.t.} \ \sum_{i=1}^{n} x_i \leq 1 \tag{3.23}$$

such that if x' is the optimal solution to this formulation, then parameter p is given by $p = (f_1(x') + f_2(x'))/2$.

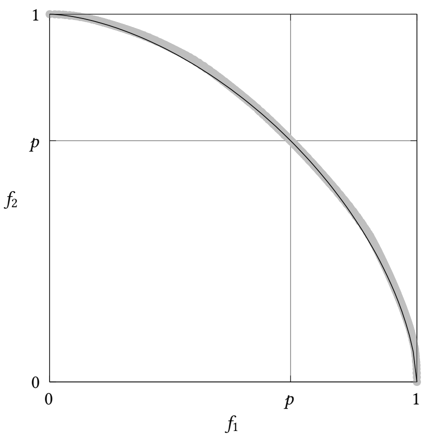
In Figures 3.10 and 3.11 we compare the performance trace given by our theoretical model against the ideal performance trace for the instances described above. We consider a varying number of linear segments for our theoretical model, and two distinct settings for the reference point. The hypervolume of the collected points is given relative to the hypervolume of the non-dominated set. We observe that the theoretical model gives a very good approximation of anytime performance for $\ell = 10$ and $\ell = 100$ linear segments, with the latter showing a slightly better prediction for non-dominated sets that can be approximated by a non-convex curve. We also see that for $\ell = 2$ linear segments, the theoretical model can only adequately estimate the ideal performance trace when the non-dominated more closely resembles a line, and the reference point is further away from the non-dominated set. Lastly, note that the model generally underestimates the hypervolume for the convex case, and overestimates it for the non-convex case.



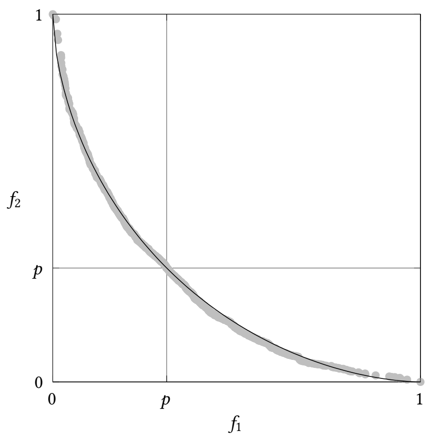
(a) $\rho = -0.8$



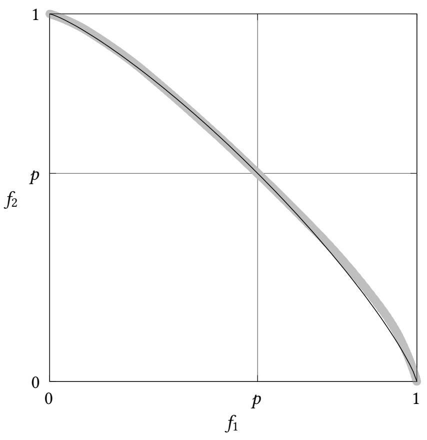
(d) $s = 1.6$



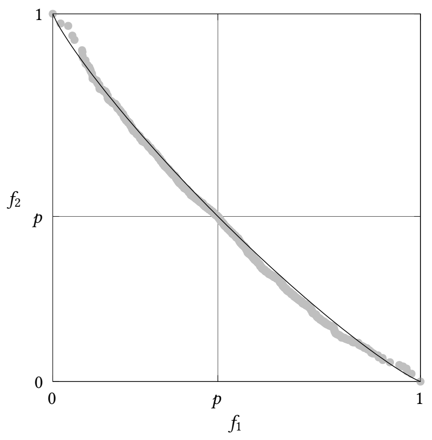
(b) $\rho = 0.0$



(e) $s = 2.0$

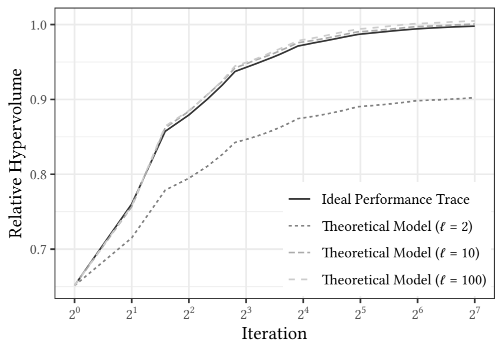


(c) $\rho = 0.8$

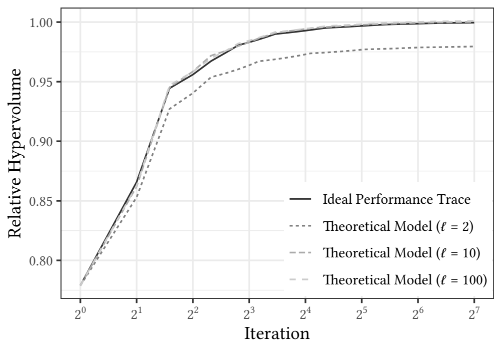


(f) $s = 3.0$

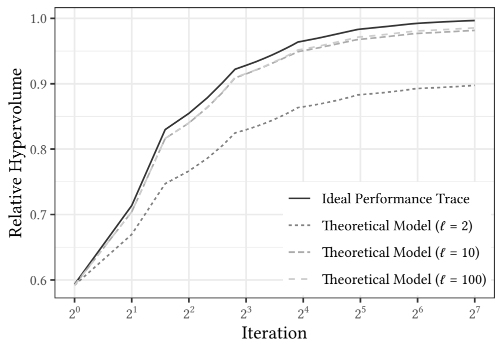
Figure 3.9: Non-dominated set (points in gray) and respective superellipse approximation (straight black line) for instances of the UBKP with varying correlation ρ and problem size $n = 100$ (left), and for instances of the CCUBKP with varying shape parameter s and problem size $n = 2000$ (right).



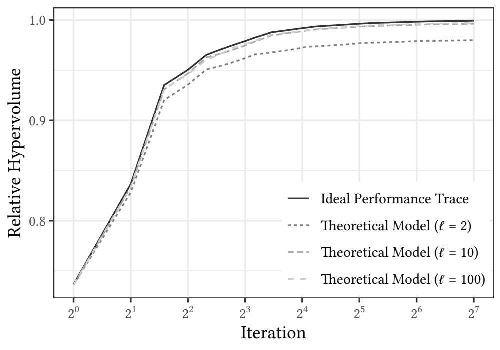
(a) $\rho = -0.8$, $r = (0, 0)$



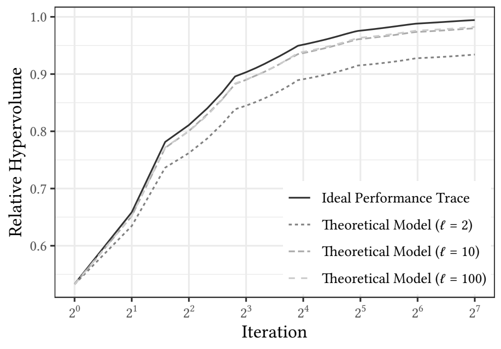
(d) $\rho = -0.8$, $r = (-1, -1)$



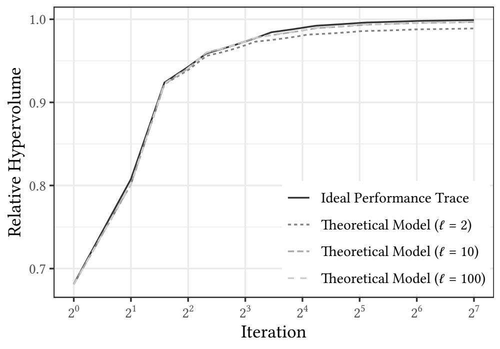
(b) $\rho = 0.0$, $r = (0, 0)$



(e) $\rho = 0.0$, $r = (-1, -1)$

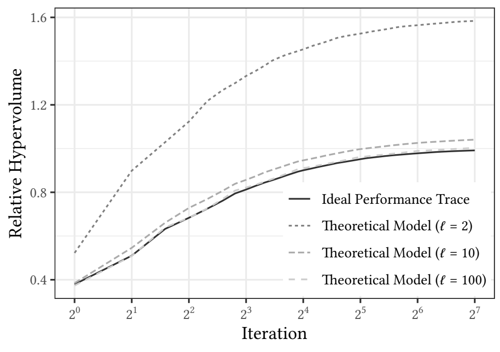


(c) $\rho = 0.8$, $r = (0, 0)$

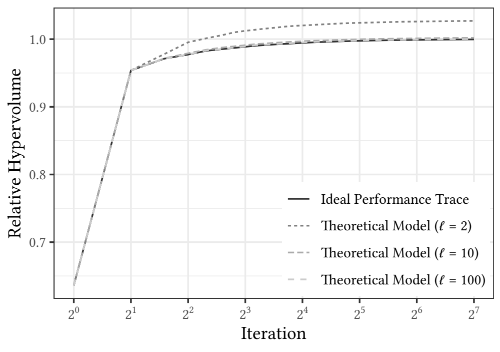


(f) $\rho = 0.8$, $r = (-1, -1)$

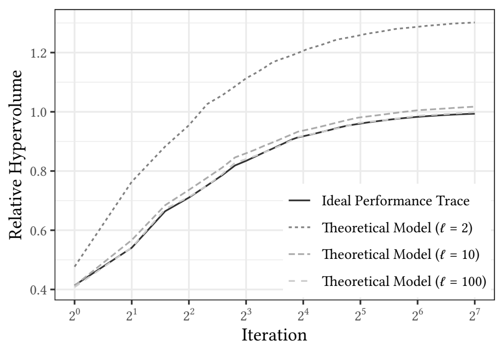
Figure 3.10: Results for the theoretical model on non-dominated sets of the UBKP with $n = 100$, reference point $r \in \{(0, 0), (-1, -1)\}$, and correlation $\rho \in \{-0.8, 0.0, 0.8\}$.



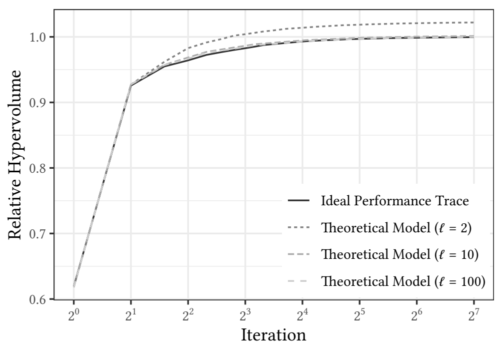
(a) $s = 1.6$, $r = (0, 0)$



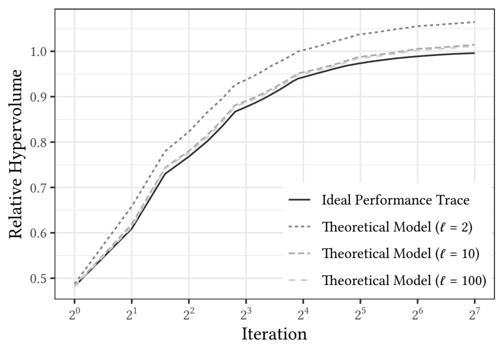
(d) $s = 1.6$, $r = (-1, -1)$



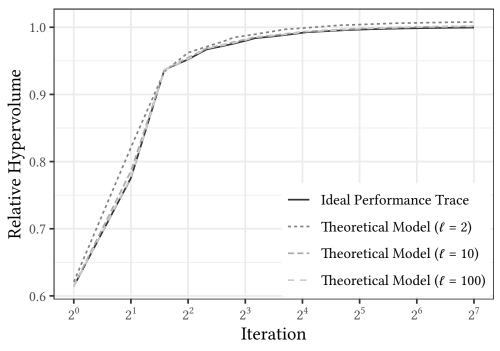
(b) $s = 2.0$, $r = (0, 0)$



(e) $s = 2.0$, $r = (-1, -1)$



(c) $s = 3.0$, $r = (0, 0)$



(f) $s = 3.0$, $r = (-1, -1)$

Figure 3.11: Results for the theoretical model on non-dominated sets of the CCUBKP with $n = 2000$, reference point $r \in \{(0, 0), (-1, -1)\}$, and shape parameter $s \in \{1.6, 2.0, 3.0\}$.

3.2 Guided ε-constraint

In this section, we present an ε-constraint approach guided by our theoretical model, namely the Guided ε-constraint (GEPS) method. As discussed in Section 2.3.1 an ε-constraint approach solves a sequence of constrained single-objective problems to find efficient solutions.

To find an efficient solution, at iteration $i > 0$, to a bi-objective optimization problem with maximizing objective functions, our approach first solves the following single-objective problem:

$$\begin{aligned} \operatorname*{argmax}_{x \in \mathcal{X}} \ & f_1(x) \\ \text{s.t. } & f_2(x) > \varepsilon^i \end{aligned} \tag{3.24}$$

where ε^i denotes a constraint on the second objective for iteration i. Note that, the problem in Equation (3.24) may yield a weakly efficient solution [16], in particular there may be a solution that gives an equal value for first objective but a better value the second. As such, to guarantee that we find an efficient solution, we consider a second problem at each iteration, which is given by:

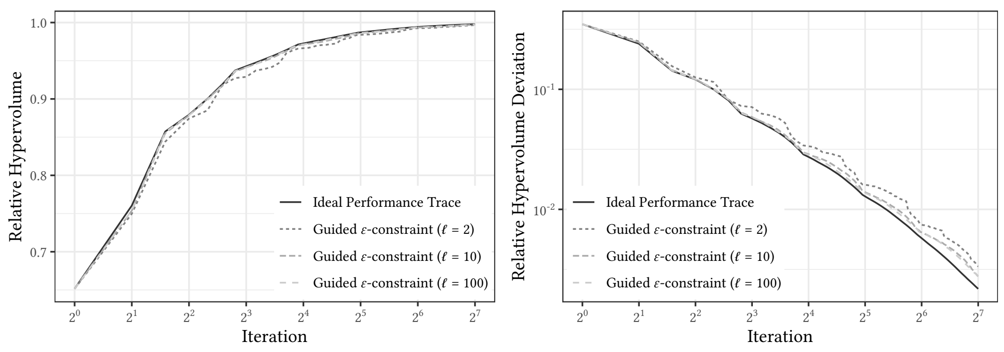
$$\begin{aligned} \operatorname*{argmax}_{x \in \mathcal{X}} \ & f_2(x) \\ \text{s.t. } & f_1(x) \geq f_1(x') \end{aligned} \tag{3.25}$$

where $x' \in \mathcal{X}$ denotes the optimal solution found for the problem of Equation (3.24).

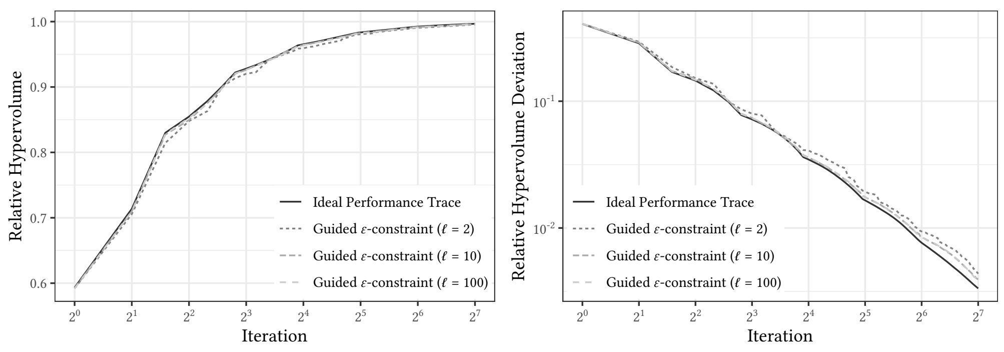
To set the ε^i constraint at iteration $i > 0$, we start by considering the second coordinate of the i-th point returned by our theoretical model. However, if it is known that no new non-dominated point can be found for that constraint, then, we consider the first subsequent point collected by our theoretical model for which a new point may be found. In particular, let y_2^i denote the second coordinate of the non-dominated point used at iteration i to set a constraint ε^i. Then, we know that no new non-dominated point can be found for a constraint between y_2^i and ε^i.

In the following, we report the performance trace for this algorithm against the ideal anytime performance. We use the GNU Linear Programming Kit MILP Solver [53] to solve the single-objective subproblems given by Equations (3.24) and (3.25) for both the UBKP and the CCUBKP. For the theoretical model used to guide the algorithm, we consider the greedy algorithmic methodology with a varying number of linear segments $\ell \in \{2, 10, 100\}$ since it returns at most one point at each iteration. Note that, if the assumptions for the analytical methodology are met, the greedy algorithmic methodology gives the same result as the analytical methodology. The performance trace is given with respect to the number of iterations and archive quality expressed in terms of relative hypervolume, i.e., the ratio between the hypervolume of the archive and the hypervolume of the non-dominated set. We also report archive quality in terms of relative hypervolume deviation, that is, the different between the maximal relative hypervolume, i.e., 1, and the actual relative hypervolume.

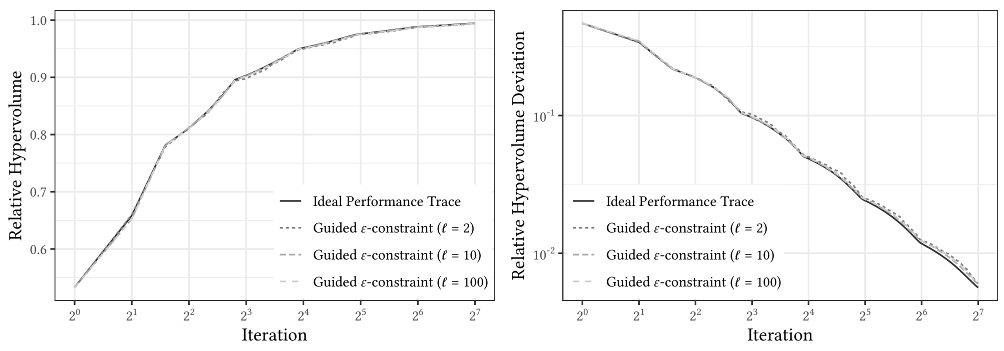
In Figures 3.12 and 3.13 we show the results for the first 128 iterations on instances for the UBKP with problem size $n = 100$, correlation values $\rho \in \{-0.8, 0.0, 0.8\}$, and reference point $r \in \{(0, 0), (-1, -1)\}$. The corresponding non-dominated sets are shown in Figures 3.9a and 3.9c. In Figures 3.14 and 3.15 we show the results on instances for the CCUBKP for a problem size $n = 2000$, varying shape parameter $s \in \{1.6, 2.0, 3.0\}$, and reference point $r \in \{(0, 0), (-1, -1)\}$. The corresponding non-dominated sets are shown in Figures 3.9d and 3.9f. Looking at the results, we can see that all three settings for the number of linear segments are quite good, and the GEPS achieves an anytime performance trace very close to the ideal one. Nonetheless, using 10 or 100 linear segments seems to provide better results, which is to be expected since the non-dominated set can be better approximated by the piecewise approximation in those cases. We observed that these results were coherent for instances with different problem sizes, different parameters, and different reference points. Moreover, we expect the results to generalize for any problem where the scaled non-dominated set can be approximated by the quadrant of a superellipse, such as the MOBKP.



(a) $\rho = -0.8$, $r = (0, 0)$

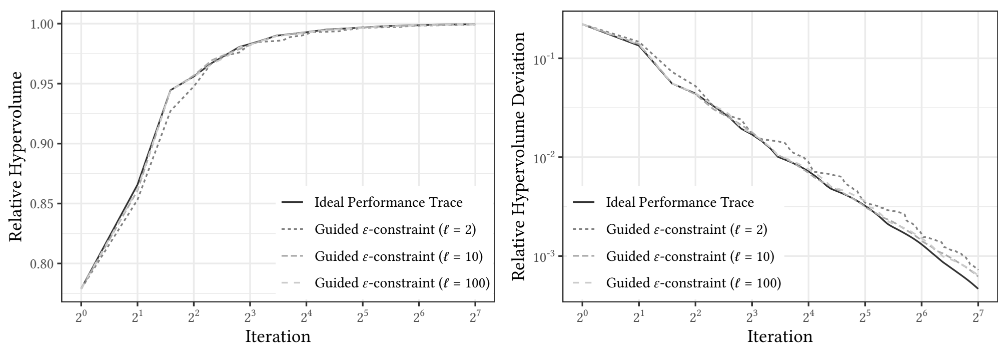


(b) $\rho = 0.0$, $r = (0, 0)$

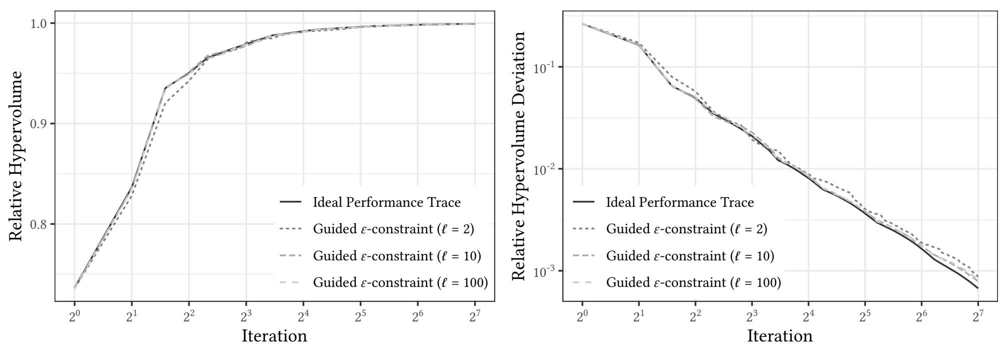


(c) $\rho = 0.8$, $r = (0, 0)$

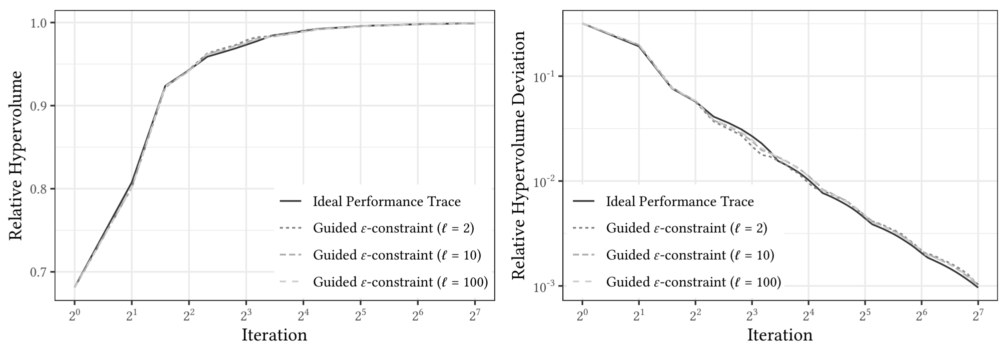
Figure 3.12: Results for the UBKP with $n = 100$, reference point $r = (0, 0)$, and varying correlation $\rho \in \{-0.8, 0.0, 0.8\}$.



(a) $\rho = -0.8$, $r = (-1, -1)$

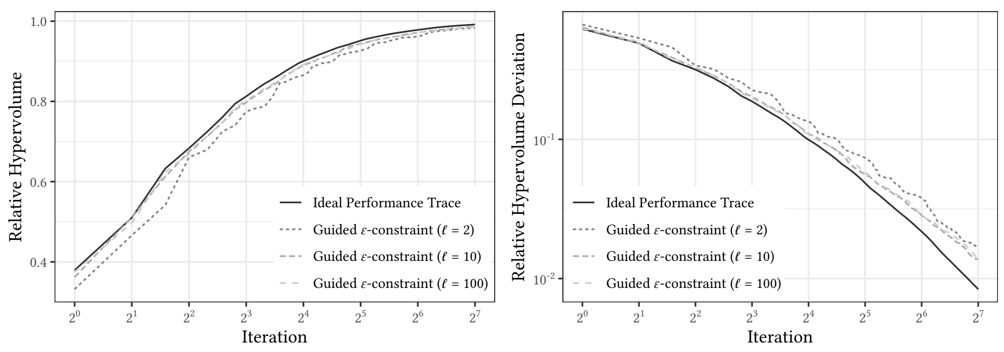


(b) $\rho = 0.0$, $r = (-1, -1)$

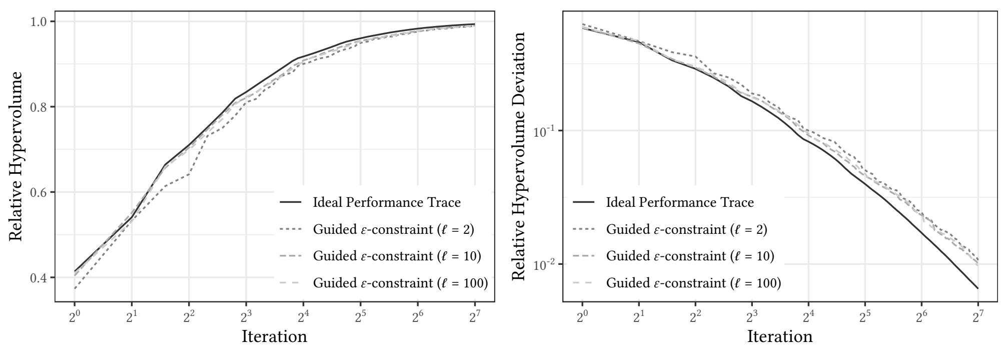


(c) $\rho = 0.8$, $r = (-1, -1)$

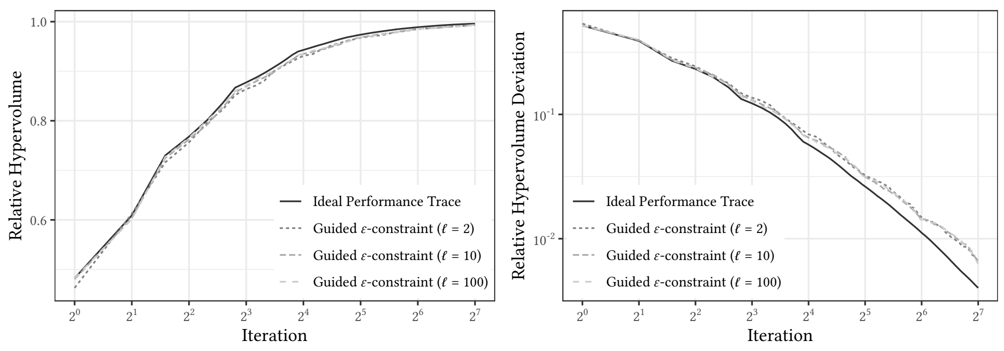
Figure 3.13: Results for the UBKP with $n = 100$, reference point $r = (-1, -1)$, and varying correlation $\rho \in \{-0.8, 0.0, 0.8\}$.



(a) $s = 1.6$, $r = (0, 0)$

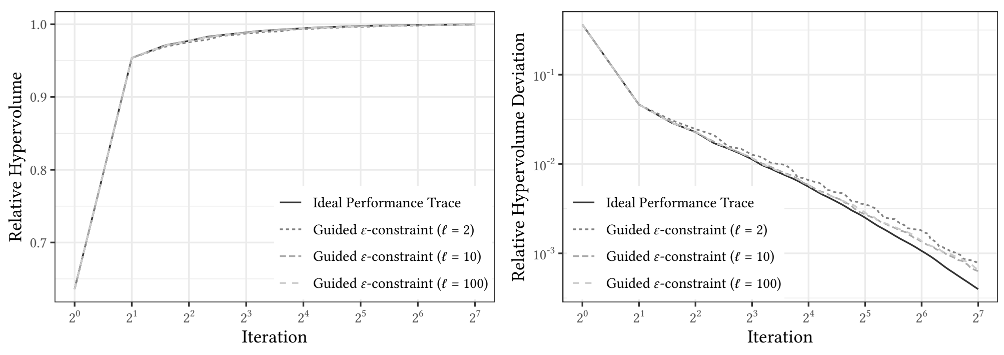


(b) $s = 2.0$, $r = (0, 0)$

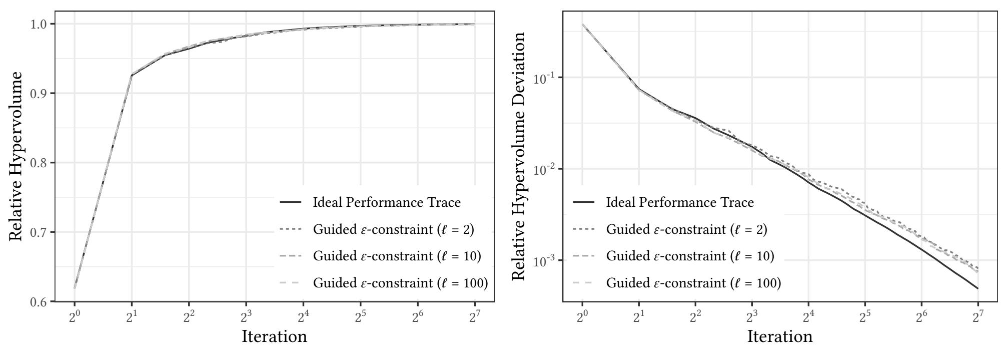


(c) $s = 3.0$, $r = (0, 0)$

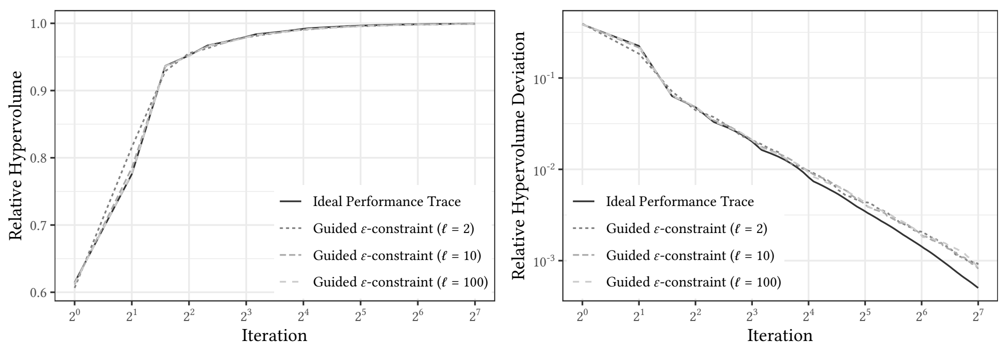
Figure 3.14: Results for the CCUBKP with $n = 2000$, capacity constraint $c = 1$, reference point $r = (0, 0)$, and varying shape parameter $s \in \{1.6, 2.0, 3.0\}$.



(a) $s = 1.6$, $r = (-1, -1)$



(b) $s = 2.0$, $r = (-1, -1)$



(c) $s = 3.0$, $r = (-1, -1)$

Figure 3.15: Results for the CCUBKP with $n = 2000$, capacity constraint $c = 1$, reference point $r = (-1, -1)$, and varying shape parameter $s \in \{1.6, 2.0, 3.0\}$.

3.3 Discussion

In this chapter, we presented a theoretical model to characterize the performance trace given by the trade-off between the number of iterations and the hypervolume for bi-objective optimization algorithms that find, at each iteration, a solution to the problem, such as scalarization techniques. In the experimental studies carried out, we observed that our theoretical model approximates quite well the ideal anytime performance. We also presented an ε-constraint approach guided by our theoretical model, namely the GEPS, which showed a performance trace very close to the ideal one.

As evidenced by this algorithm, a possible use of our model is to design scalarization techniques with good anytime performance. Moreover, our model can be used to monitor and predict the anytime performance of our algorithm or techniques that similarly attempt to maximize the hypervolume at each iteration [61]. For example, it should be possible to detect if an algorithm that has an anytime performance similar to our model is taking too long to find each solution, and whether or not it will achieve some desirable quality within a certain time budget, which may justify a restart or a switch to a different strategy.

One possible direction for future research, is to extend the model for approximations of the non-dominated set other than a quadrant of a superellipse. In practice, this can already be accomplished by our algorithmic methodology by considering any set of mutually non-dominated, possibly disconnected, linear segments. However, the main challenge is to define a good piecewise linear approximation to the non-dominated set. A related idea is to consider a feedback loop between our model and a scalarization technique that sequentially finds efficient solutions. In particular, if the set of linear segments corresponds to an approximation of the non-dominated set, and throughout the execution of the scalarization technique we find the actual non-dominated points, then we can insert these points back into the model to increase the quality of the approximation in an online fashion, which could be used to better predict where the next points in the non-dominated set will be. Lastly, a clear direction for future research is to extend our model for more than 2 objectives. This seems more challenging in part due to the splitting and tracking of the uncovered regions after a point with maximal hypervolume contribution is found, since the regions may no longer be independent, i.e., finding a point in one region may affect the other regions.

Lastly, for the purposes of algorithm selection in this thesis, it would be more useful to have a model that considers anytime performance in terms of

CPU-time rather than the number of iterations. In the next chapter, we discuss an empirical model to convert the number of iterations given by this theoretical model to CPU-time by considering empirically collected data.

Chapter 4

Empirical Models of Anytime Performance

In this chapter, we consider the development of empirical models for predicting the anytime performance of an algorithm, which take into account empirically collected anytime performance data from previous runs of the algorithm. Unlike the theoretical model discussed in the previous chapter, these empirical models can be used to predict anytime performance regardless of time or quality unit, as long as the anytime performance data to train the models is collected in terms of the desired units of time and quality. In particular, this is desirable for the algorithm selection methodologies that we will be considering in the following chapters, as we want to consider time in terms of CPU-time, which was not possible with the theoretical model of the previous chapter.

This chapter is organized as follows. In Section 4.1 we propose three abstract frameworks for the development of empirical models that take into account anytime performance traces collected for an algorithm on a set of training instances. Then, in Section 4.2 we propose an empirical model for predicting the anytime performance of algorithms to the MOBKP that follows the first of these proposed frameworks. We also carry out an experimental study considering three algorithms and instances with 2, 3, and 5 objectives, which shows that the model can often predict the anytime performance correctly. However, we also highlight some cases for which the model is not as accurate as we would hope, and we discuss possible reasons for this. Lastly, in Section 4.3 we summarize the main results of this chapter and discuss possible directions for future work.

4.1 Frameworks for Empirical Models

In this section we propose three abstract frameworks to guide the development of empirical models of anytime performance. We recall that we want to develop empirical models with the goal of predicting the anytime performance of an algorithm on a previously unseen problem instance. In particular, let ι denote a previously unseen problem instance, and let a denote an algorithm to solve ι. We consider that an empirical model to predict the performance of algorithm a on instance ι can be characterized in terms of a performance profile:

$$\widetilde{P}_{a,\iota}(t,q) \to [0,1] \tag{4.1}$$

that denotes the (approximate) likelihood of algorithm a achieving an approximation with quality greater than or equal to q at time t, when solving problem instance ι.

In all the presented frameworks, we assume that there is a set of training instances $\mathcal{I}$ that is known, and that there is a set of runs for algorithm a associated to this set of instances, denoted by $R_{a,\mathcal{I}}$. We generally assume that the instances on the training data are not too different from the instances for which we will be predicting anytime performance. In particular, we assume that when predicting the anytime performance for an instance ι, that there is a set of instances in the training data set that give an anytime performance similar to that of instance ι. This is not to say that we cannot predict anytime performance if there are no such instances in the training data set. However, we expect the quality of the prediction to be better if there are.

Another relevant aspect regarding the training data set is that in the presented frameworks the training set may be updated at any time. This may be particularly relevant in real-world scenarios where one possibility to build the training data set is to continuously update the training data set with anytime traces from previously encountered instances.

Lastly, these frameworks are discussed in general terms, since the implementation details for the model can depend on the prediction scenario, e.g., how much human-interaction is possible and what information can be gathered from the problem instances. That being said, an empirical model to predict the anytime performance of algorithms to the MOBKP problem, following the first framework, is presented in Section 4.2.3.

4.1.1 Framework I

Given the set of training instances $\mathcal{I}$ and a previously unseen instance $\iota \notin \mathcal{I}$, a model following this framework starts by selecting a subset of training instances, denoted by $\mathcal{I}_S \subseteq \mathcal{I}$ that are expected to have similar anytime performance to instance ι. Then, given a set of runs $R_{a,\mathcal{I}_S}$ denoting the runs of algorithm a on the selected training instances $\mathcal{I}_S$, the model builds and returns an approximated performance profile given by the empirical performance profile for those runs, that is:

$$\widetilde{P}^{\mathrm{I}}_{a,\iota}(t,q) = \widehat{P}_{R_{a,\mathcal{I}_S}}(t,q) \tag{4.2}$$

The key aspect for this model is how to select a subset of training instances $\mathcal{I}_S$ that are expected to have similar anytime performance to the previously unseen instance ι. One possibility is to manually design a set of rules to perform this selection, based on a set of instance features that characterize a problem instance. This requires that the behavior of algorithm a is well known, and that it is obvious how the different instance features affect anytime performance, which, arguably, is not often the case.

Another possibility, that requires less human interaction, is to manually analyze the anytime performance of an algorithm on the training instances, and to group instances that give similar anytime performance. Then, given a set of instance features that characterize an instance, we can use classification techniques to automatically learn to classify the group of instances to which the previously unseen instance ι belongs. Alternatively, given a set of measurable features that characterize the anytime performance of an algorithm on an instance, we could also build the groups without any human interaction by using an unsupervised partitioning technique that takes into account those features, such as k-means clustering.

Lastly, another possibility is to consider automatic learning techniques that can learn to select similar instances based on a set of instance features without the need for human interaction. For example, given a set of instance features that characterize a problem instance, we can use clustering techniques, such as a k-nearest neighbor algorithm, that can select instances that have similar instance features, and which we expect to have similar anytime performance. This is the possibility that we consider later in Section 4.2.3 when implementing our model.

4.1.2 Framework II

To predict the approximation quality at time t with the previous framework, we assume that the runs of an algorithm on the training instance set were interrupted at a time greater than or equal to t, or that the algorithm finished before time t, since otherwise there is no empirical data that can be used to build the predicted performance profile. This can lead to infeasible training times if we want to predict the approximation quality for large values of t. To address this, our second framework considers the use of parametric models that characterize the anytime performance of an algorithm on the training instances $\mathcal{I}$, to predict the anytime performance on a previously unseen instance $\iota \notin \mathcal{I}$.

In the field of statistics, data collected over time for the same subject, in this case the performance trace of a run, is commonly known as *longitudinal data* [20]. An issue posed by longitudinal data is that measurements taken from the same subject are not independent. To address this, *mixed-effects models* where each subject is modeled by a separate curve whose parameters are a random perturbation of a baseline curve can be considered [20]. Note that, since the performance traces of an anytime algorithm are not typically expected to be linear, a non-linear mixed-effects model is likely to be more appropriate. Assuming that the mean response of a subject i at time t_{ij} can be modeled in terms of a non-linear regression function and random error, a non-linear mixed effects model is given by:

$$y_{ij} = f(t_{ij}, \beta_i) + \epsilon_{ij} \tag{4.3}$$

where the evolution of the mean response y_{ij} is modeled by a non-linear function $f(t_{ij}, \beta_i)$ and a random error term ϵ_{ij}. The non-linear function $f(\beta_i, t_{ij})$ takes the time parameter t_{ij} and a set of subject-specific parameters β_i. These parameters β_i are commonly defined to linearly depend on a set of covariates A_i, and random effects b_i, that is:

$$\beta_i = A_i\beta + b_i \tag{4.4}$$

Alternatively, parameters β_i can also be modeled by a non-linear function. We refer to [20] and [63] for a more complete description and discussion of non-linear mixed effects models.

In the context of anytime algorithms, Gagliolo et al. [21] considered non-linear mixed-effects models following an exponential decay growth curve to model the anytime performance of a stochastic single-objective optimization algorithm and predict the optimum objective value, given a set of 400 instances and 25 runs of the algorithm for each instance. There were two experiments that defined subjects for the non-linear mixed effects models differently. In the first, the authors fit a separate model for each instance, taking into account the 25 runs gathered for each instance as the subjects in the model. In the second experiment, the authors fit 25 models, such that for each model a single run for each instance is chosen, and the model takes into account the instances as the subjects.

For the development of an empirical model under this second framework, we consider two possibilities to develop the parametric model. For the first possibility, we consider the idea of the first experiment by Gagliolo et al. [21], and fit a separate model for each training instance based on the available runs. Since each separate model is for a single instance, instance specific features are not mapped to the response by the model itself. As such, we need to learn to predict the parameters β_i of the non-linear function that were fit by the model from a set of instance features, e.g., by using supervised learning techniques from machine learning. Then, given a previously unseen instance $\iota \notin \mathcal{I}$, we map its instance features to a set of predicted parameters $\widetilde{\beta_\iota}$ to be considered in the non-linear function. The estimated performance trace for an algorithm a on a previously unseen instance ι is given by:

$$\widetilde{P}^{\text{II}}_{a,\iota}(t, q) = \begin{cases} 1 & \text{if } f_a(t, \widetilde{\beta_\iota}) \geq q \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

where $f_a(t, \widetilde{\beta_\iota})$ is the non-linear function considered for algorithm a that takes as parameters the time t and the predicted set of parameters $\widetilde{\beta_\iota}$. If, instead, multiple non-linear mixed-effects models were fit for each instance, e.g., a set of non-linear mixed-effects models fit using quantile regression methods [24], then we could learn to predict the parameters from each model and build the estimated performance profile from multiple curves.

The second possibility we consider is to fit a non-linear mixed-effects model that takes into account the instances as subjects, e.g., by considering the median performance trace for each instance. In this case, instance features can be given as parameters to the non-linear function. Then, when given a previously unseen instance $\iota \notin \mathcal{I}$ we can apply the covariates A_i that were fit by the model to the instance features of ι to get the predicted median anytime performance trace for ι. As before, we can consider different models for different quantiles to have a more complete anytime performance profile. We expect that it is harder to build a model following this second possibility, since a function needs to be defined to map the instance features to the non-linear function parameters, which may not be trivial in many cases. However, it may allow to better predict instances that are quite different from the training instances.

4.1.3 Framework III

This frameworks assumes a theoretical model that characterizes the performance trace of an algorithm on a previously unseen instance with respect to the number of iterations and archive quality, such as the one presented in Section 3.1. In particular, let this theoretical model be denoted by a function $\widetilde{Q}_{a,\iota}(i) \to \mathbb{R}$ that returns the predicted quality of the archive at iteration $i \in \mathbb{Z}_{>0}$

for an algorithm a on instance ι. Moreover, let $\mathcal{I}_S \subseteq \mathcal{I}$ denote a set of selected training instances that, compared to instance ι, take a similar amount of CPU-time going from iteration i to $i+1$ for every $i \in \mathbb{Z}_{\geq 0}$. Lastly, let $R_{a,\mathcal{I}_S}$ denote the runs of algorithm a on the training instances $\mathcal{I}_S$, and $T_{a,r}(i) \longrightarrow \mathbb{R}_{\geq 0}$ denote a function that returns the time algorithm a took to reach iteration i on run r.

Then, we define a function:

$$\widetilde{Q}_{a,\iota,r}(t) = \widetilde{Q}_{a,\iota}\left(\min\{i : T_{a,r}(i) \geq t\}\right) \tag{4.6}$$

that denotes an approximate performance trace for instance ι assuming the CPU-times obtained for run r. Finally, the approximate performance profile obtained by this framework is given by:

$$\widetilde{P}^{\mathrm{III}}_{a,\iota}(t,q) = \frac{1}{|R_{a,\mathcal{I}_S}|} \sum_{r \in R_{a,\mathcal{I}_S}} \begin{cases} 1 & \text{if } \widetilde{Q}_{a,\iota,r}(t) \geq q \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

The key aspect for this framework is how to select training instances that take a similar amount of time on each iteration, compared to a previously unseen instance ι. As in Framework I, we can either consider a set of rules designed by hand, or use (semi)-automated learning techniques to identify similar instances. However, one important difference is that for this model we need to take into account what happens at each iteration. For example, consider the GEPS approach presented in Section 3.2, which can be modeled by the theoretical model presented in Chapter 3. For an iteration to take a similar amount of time when solving two distinct instances, the scalarized ε-constraint problems to be solved should have similar difficulty so that they take a similar amount of time to solve. As such, an approach following this empirical model should look at instance features that affect the performance of the solver to the scalarized problem, e.g., how tight the ε^i constraint is at each iteration.

4.2 Experimental Study

In this section, we carry out an experimental study for an empirical model implemented according to Framework I. In particular, we consider the use of this model in predicting the anytime performance of PLS, BHV-DP, and GEPS algorithms on the MOBKP considering different numbers of objectives. This section is organized as follows. In Section 4.2.1, we describe the procedure used to generate the problem instances used in this study. In Section 4.2.3, we give the implementation details of the model. In Section 4.2.4, we describe the setup used for the experimental study. And lastly, in Section 4.2.5, we present and discuss the results of this study.

4.2.1 Instances

Benchmark instances for the MOBKP are generated according to the following procedure. For a given number of items $n \in \mathbb{Z}_{>0}$ and objectives $m \in \mathbb{Z}_{\geq 2}$, the item values $v_i^j \in [0,1]$, $i \in \{1,\dots,n\}$, $j \in \{1,\dots,m\}$, are generated according to a multi-variate uniform distribution of dimension m, defined by a positive-definite symmetric correlation matrix C of size $m \times m$, such that $C_{aa} = 1$ and $C_{ab} = \rho_v$ for all $a, b \in \{1,\dots,m\}$ with $a \neq b$, that is

$$\begin{pmatrix} 1 & \rho_v & \cdots & \rho_v \\ \rho_v & 1 & \cdots & \rho_v \\ \vdots & \vdots & \ddots & \vdots \\ \rho_v & \rho_v & \cdots & 1 \end{pmatrix}$$

where $\frac{6}{\pi} \sin^{-1} \frac{1}{2(1-m)} < \rho_v < 1$ denotes the correlation between any two distinct value vectors v^j and v^k. We follow the procedure described by [73] to generate the value vectors under these conditions. In particular, we start by generating vectors $(V^1,\dots,V^m)$ following a multi-normal distribution of means 0 and correlation matrix $R = 2\sin(\frac{\pi}{6}C)$. Then, the vectors $v^1 = \phi(V^1),\dots,v^m = \phi(V^m)$, such that $\phi(\cdot)$ is the univariate normal cumulative density function, are uniformly distributed with a correlation matrix C.

The weight vector w is generated following a uniform distribution in the range $[0,1]$ with approximate correlation ρ_w to the auxiliary vector

$$v' = \left(\sum_{j=1}^{m} v_1^j, \dots, \sum_{j=1}^{m} v_n^j\right) \tag{4.8}$$

that gives the sum of the values for each item. Let $z(v')$ denote a function that centers and scales v' to have mean 0 and standard deviation 1, and let $E \sim \mathcal{N}(0,1)$ be a vector of size n drawn from a normal distribution with mean 0 and standard deviation 1. Then, we consider the following formulation to get an auxiliary vector w' with correlation approximately ρ_w:

$$w' = \rho_w z(v') + E\sqrt{1-\rho_w{}^2} \tag{4.9}$$

Note that, w' is not drawn from a population with uniform distribution. However, we can run a Monte Carlo simulation to repeatedly resample from the same population as w' and get an empirical cumulative density function. Then, we can get a vector w drawn from a approximate uniform distribution with:

$$w = \widetilde{\phi}(w') \tag{4.10}$$

where $\widetilde{\phi}(w')$ denotes the empirical cumulative density function of the population of w'. The correlation between w and v' is approximately ρ_w.

Lastly, the weight constraint W is defined as a ratio of the sum of the items weights, with respect to a variable $\omega \in [0, 1]$ as follows:

$$W = \omega \sum_{i=1}^{n} w_i \tag{4.11}$$

We remark that the commonly used benchmark instance classes described by Bazgan et al. [3] for two objectives, can be generalized for two and more objective using the procedure described above, in particular:

A. Uncorrelated instances: $\rho_v = 0$, $\rho_w = 0$, and $\omega = 0.5$.

B. Unconflicting instances: ρ_v is set to a positive value, $\rho_w = 0$, and $\omega = 0.5$.

C. Conflicting instances: ρ_v is set to a negative value, $\rho_w = 0$, and $\omega = 0.5$.

D. Conflicting instances with correlated weight: ρ_v is set to a negative value, ρ_w is set to a positive value, and $\omega = 0.5$.

For this experimental study, we will be considering instances generated with the following parameters:

- $m \in \{2, 3, 5\}$
- $n \in \mathcal{U}\,(50, 150)$
- $p_v \in \mathcal{U}\left(\frac{6}{\pi}\sin^{-1}\frac{1}{2(1-m)}, 1\right)$
- $p_w \in \mathcal{U}\,(-1, 1)$
- $\omega \in \mathcal{U}\,(0.3, 0.7)$

where $\mathcal{U}(\cdot)$ denotes the uniform distribution.

4.2.2 Anytime Performance Measure

To measure the performance of an empirical anytime performance profile P as a scalar value, we consider an *anytime performance measure* given by:

$$M(P) = \int_{q_\ell}^{q_u} \int_{0}^{t_u} P(t, q)\, dt\, dq \tag{4.12}$$

where q_ℓ and q_u denote the lower and upper bound for solution quality, and t_u denotes the upper bound on time. This is a particular case of the anytime performance measure given by Jesus et al. [36], and related to the anytime performance measure given by López-Ibáñez and Stützle [51]. The former will be presented in more detail in Section 5.1.

4.2.3 Model Implementation Details

In this section we describe the implementation details of our model, which follows Framework I. We recall that according to this framework, the goal is to select a subset of problem instances from the training data set $\mathcal{I}$ that are expected to have similar anytime performance to the previously unseen problem instance $\iota \notin \mathcal{I}$ for which we want to predict anytime performance. Then, we take the performance traces collected a priori for the selected instances, and use them to build a performance profile that predicts the anytime performance of the previously unseen instance ι.

To define the implementation of our model, we consider a training data set of problem instances to the MOBKP for which an anytime performance trace resulting from a single run has been collected for each algorithm. We also consider the following set of instance features that characterize a problem instance for the MOBKP as a set of measurable values:

- $\mathcal{F}_1$ — the number of decision variables n;
- $\mathcal{F}_2$ — the ratio between the capacity of the knapsack W, and the sum of the items' weights, i.e.:
$$\mathcal{F}_2 = \frac{W}{\sum_{i=1}^{n} w_i} \tag{4.13}$$
where w_i denotes the weight of item $1 \leq i \leq n$;
- $\mathcal{F}_3$ — the mean Spearman's correlation coefficient between all pairs of value vectors v^i, $1 \leq i \leq m$, i.e.:
$$\mathcal{F}_3 = \frac{1}{m(m-1)/2} \sum_{i=1}^{m} \sum_{j=i+1}^{m} r_{v^i v^j} \tag{4.14}$$
where $r_{v^i v^j}$ denotes the Spearman's correlation coefficient between value vectors v^i and v^j. This feature relates to parameter ρ_v that was used to generate the problem instance;
- $\mathcal{F}_4$ — the Spearman's correlation coefficient between the weights of the items w, and the sum of their values, i.e., $\sum_{i=1}^{m} v^i$. This feature relates to parameter ρ_w that was used to generate the problem instance.

The empirical model then follows a k-nearest neighbor algorithm. In particular, we compute the distance between the features of the problem instances in the training data set and the features of the previously unseen instance ι. The distance between features is considered with respect to the weighted Euclidean norm, i.e.:

$$\left\|\mathcal{F}^\iota - \mathcal{F}^j\right\|_\xi = \sqrt{\sum_{\ell=1}^{4} \left(\mathcal{F}_k^\iota - \mathcal{F}_k^j\right)^2 \xi_k} \tag{4.15}$$

where $\xi_\ell > 0$ denotes the weight for the ℓ-th feature, $\mathcal{F}^\iota$ denotes the feature set of the problem instance ι whose anytime performance we want to model, and $\mathcal{F}^j$ denotes the feature set of a problem instance j in the training data set. To make it easier to reason about the weights we scale all features to the range $[0, 1]$ taking into account their range of possible values, i.e.:

- $\mathcal{F}_1 \in [50, 150]$
- $\mathcal{F}_2 \in [0.3, 0.7]$
- $\mathcal{F}_3 \in [-1/(m-1), 1]$
- $\mathcal{F}_4 \in [-1, 1]$

To define the weights ξ for the distance metric, we consider the Spearman's correlation coefficient between the features and the anytime performance measure for the instances. In particular, let the $\rho_{\mathcal{F}_\ell}$ denote the Spearman's correlation coefficient for the instance feature $\mathcal{F}_\ell$, then weight ξ_ℓ is defined as:

$$\xi_\ell = \exp(4\,|\rho_{\mathcal{F}_\ell}|) \tag{4.16}$$

We consider this definition because we want features that have a large absolute correlation value to have a significantly larger impact than features that have a small absolute correlation value. However, we do not want this difference to be too large, which is why we only multiply the absolute correlation value by 4 and not by a larger number, since features with a small absolute correlation value might still be relevant to distinguish between otherwise similar instances. Alternatively, we could set the weight values in a more informed manner, e.g., by using a grid search over possible values for the weights and taking into account the error for the model using different weights. However, this would result in much higher training times, so we opted to use the above definition instead since it gave good results in a preliminary analysis.

After computing the distances between each instance from the training data set and the previously unseen instance ι, we select the k instances from the training data set that minimize the distance. Then, we aggregate the anytime traces of the algorithm on those instances and build an anytime performance profile, which is expected to predict the anytime performance of instance ι. Note that, the range of hypervolume values for different problem instances can be very different. As such, for the aggregation to be meaningful, we consider the relative hypervolume for measuring solution quality, which means that solution is defined in the range $[0, 1]$ for all anytime traces. Since we cannot always compute the non-dominated set in a feasible amount of time to have the maximal hypervolume, we use the maximum hypervolume found by any of the algorithms we tested for a particular instance in order to compute the relative hypervolume metric for each instance.

To validate our model, we split our known problem instances into two distinct data sets: training and testing. In particular, we consider 80% of the instances for the training data set, and 20% of the instances for the testing data set. Then, we consider both numerical and visual analysis. For the former, we consider the difference between the anytime measure of the predicted anytime performance profile, and the anytime measure of the anytime trace gathered for a problem instance ι taken from the testing data set. To summarize the quality of our model over all instances in the testing data set, we consider the Mean Absolute Error (MAE), which is given by the absolute difference between the predicted and true anytime measures for each testing problem instance divided by the number of testing problem instances. For visual analysis, we plot the predicted performance profile and the true anytime trace.

Selecting an appropriate value for k in the k-nearest neighbor algorithm is of the utmost importance to the quality of our model. In particular, if the value for k is too small we are likely to select instances that are very similar in terms of the instances features we considered. However, we might not have enough anytime traces to build an anytime performance profile that is sufficiently robust against random variations on anytime performance that may arise from the stochastic nature of the algorithm, experimental conditions, or from other instance features that have not been considered. On the other hand, if the value for k is too large we might select instances that are too different from the instance whose anytime performance we want to predict, which will result in an anytime performance profile that is not specific enough.

To set parameter k for the model, we consider the MAE of our model given by the Leave-One-Out Cross-Validation (LOOCV) method for different values of k. In LOOCV we take the training data set, leave out one instance, and use the remaining instances for training our model. Then, we use the trained model to predict the anytime performance of the left-out instance. We repeat this process such that every training instance is left out once, and compute the MAE considering the predicted and true values for each instance. We compute the MAE for each value of k, considering the same set of training problem instances, and choose the value of k that gave the minimum MAE to build our final model using the full training data set.

We remark that the choice of LOOCV, rather than other resampling methods, is due to its small bias when considering a small number of samples [57]. Nonetheless, preliminary experiments considering a 10-fold cross-validation method yielded similar results. The choice of MAE, instead of Root-Mean-

Square Error (RMSE), which is also commonly used, is due to the fact that we expect that there are a small number of problem instances whose prediction will give large errors and cannot be significantly improved, and MAE is less sensitive to large errors since it does not square the errors.

The implementation of this model is available at [32].

4.2.4 Experimental Setup

To test our model we generated 500 instances for each number of objectives $m \in \{2, 3, 5\}$. For each instance, the value of n was randomly sampled from the uniform distribution $\mathcal{U}(50, 150)$, the value of ρ_v was randomly sampled from the uniform distribution $\mathcal{U}\left(\frac{6}{\pi}\sin^{-1}\frac{1}{2(1-m)}, 1\right)$, the value of ρ_w was randomly sampled from the uniform distribution $\mathcal{U}(-1, 1)$, and the value of ω was randomly sampled from the uniform distribution $\mathcal{U}(0.3, 0.7)$.

For each instance, we record the anytime performance trace, in terms of CPU-time and hypervolume, for one run of the BHV-DP and PLS algorithms for $m \in \{2, 3, 5\}$ objectives, and the GEPS algorithm for $m = 2$ objectives. As for the configuration of the algorithms, for the BHV-DP algorithm we consider the ordering of the items given by $\mathcal{O}^{\min}$ since it gave the best overall results during preliminary experiments, which is consistent with the results reported by Bazgan et al. [3]. For the PLS algorithm, we consider the default ordering of the items, an initial solution corresponding to the empty knapsack, i.e., $x_i = 0$ for all $i \in \{1, \ldots, n\}$, and a random selection of the next item to process. We chose this configuration since it often achieved a solution with good quality quickly. For the GEPS algorithm we considered a parameter $\ell = 100$, since it showed good results in Chapter 3, and in preliminary experiments for the MOBKP. The algorithms were implemented in `C++` and are available at [33].

The algorithms were executed on a computer with two Intel(R) Xeon(R) Silver 4210R CPUs with clock frequency 2.40GHz, 10 cores, and 20 threads each. The algorithms were executed with a CPU time limit of 100 seconds and a memory limit of 8 Gb. Moreover, due to the large amount of time required to run all the experiments, the executions were carried out in parallel, 20 at a time (corresponding to one execution per CPU core). Although this is likely to impact the performance of the algorithms, we consider that it does not affect the results for our models, since both the training and testing instances are generated under the same conditions. In a real-world scenario, training instances should be generated under the same conditions that the new instances are expected to be solved on. The code used to generate the problem instances, and to gather the performance traces of the algorithms, is available at [32].

4.2.5 Results

In this section we discuss the results of our model for predicting the anytime behavior of the PLS and BHV-DP algorithms for problem instances with $m \in \{2, 3, 5\}$ objectives, and of the GEPS algorithm for problem instances with $m = 2$ objectives. The code to reproduce these experiments is available at [32].

PLS — 2 Objectives

We start by analyzing the results of predicting the anytime behavior of the PLS algorithm on problem instances with 2 objectives.

Figure 4.1 gives the density plot of the anytime measure values for problem instances in the training and testing data sets. Note that, anytime measure is defined in the range $[0, 100]$ since CPU-time is measured in seconds between 0 and 100, and solution quality, given by the relative hypervolume, is defined between 0 and 1. The figure shows that both the training and testing data sets have a similar distribution. As such, we expect to be able to correctly predict anytime performance for most instances. One exception is for problem instances with a small anytime measure. Since there are not many problem instances with anytime performance below 85, we expect that by selecting a fixed number of similar instances k, that the model will inevitably choose instances that do not have similar anytime anytime behavior.

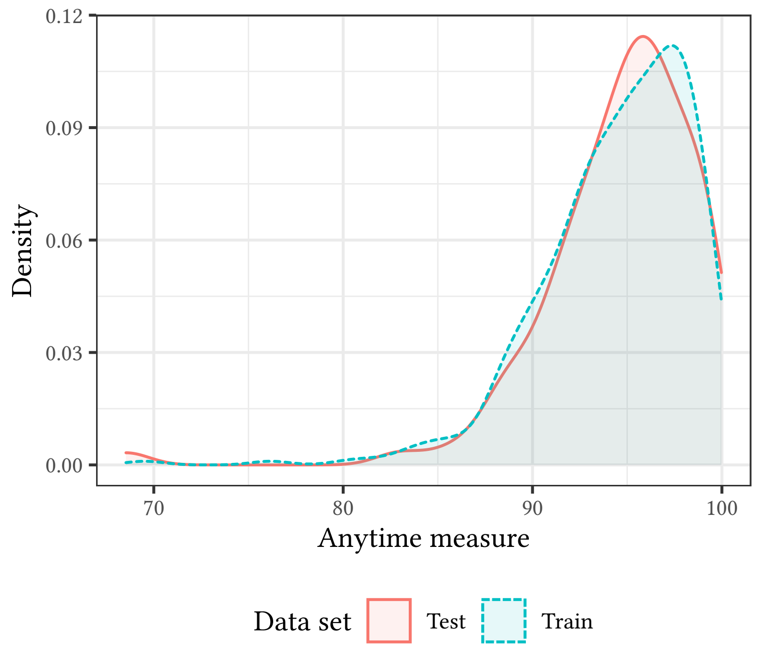


Figure 4.1: Anytime measure density plot for training and testing problem instances. PLS, 2 objectives.

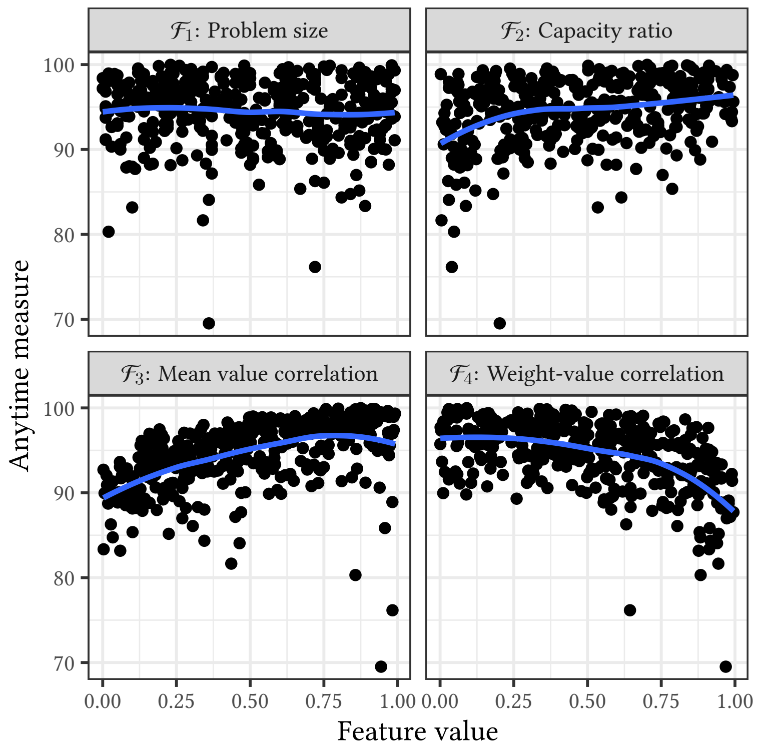


Figure 4.2: Relation between anytime measure and feature values for training problem instances. PLS, 2 objectives.

Figure 4.2 shows the relation between feature values and anytime measure for each problem instance in the training data set. The blue line, given by the Locally Estimated Scatterplot Smoothing (LOESS) method [8], highlights the relationship between the feature values and the anytime measure. Table 4.1 gives the Spearman's correlation coefficient between each feature and the anytime measure of the training instances. These results suggest that anytime measure is most impacted by the correlation features, $\mathcal{F}_3$ and $\mathcal{F}_4$. The capacity feature, $\mathcal{F}_2$, also appears to have an impact on anytime measure, but to a lesser degree. The problem size feature, $\mathcal{F}_1$, appears to have no discernible impact on anytime measure. Table 4.2 gives the weights that were set according to Equation (4.16). We recall, that a higher value of ξ_i implies that feature $\mathcal{F}_i$ has a larger impact on the computed distance between instances.

Table 4.3 gives the MAE value obtained by LOOCV on the training data

$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
-0.087	0.289	0.620	-0.504

Table 4.1: Spearman's correlation coefficient between the features and anytime measures of the training instances. PLS, 2 objectives.

ξ_1	ξ_2	ξ_3	ξ_4
1.415	3.175	11.919	7.512

Table 4.2: Weights for the distance metric. PLS, 2 objectives.

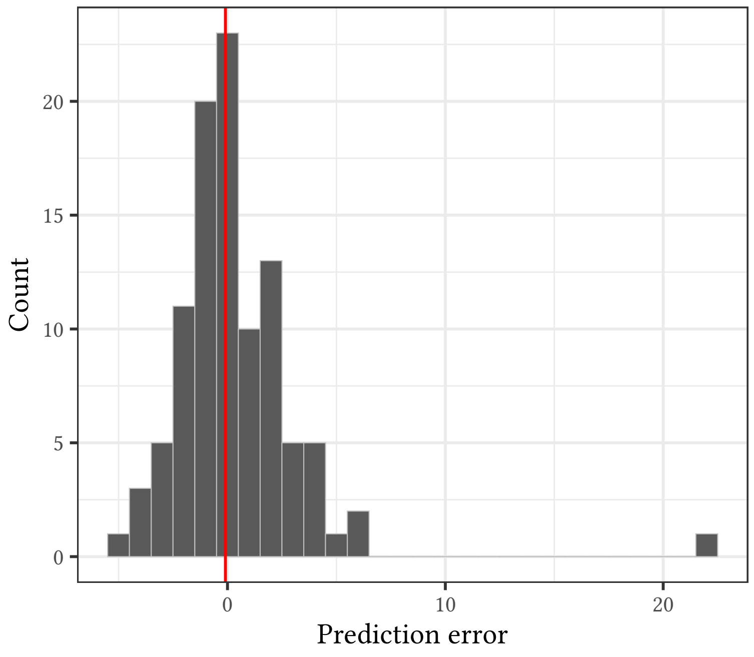


Figure 4.3: Histogram of the prediction error in terms of anytime measure for testing problem instances. The red line corresponds to the median. PLS, 2 objectives.

set for each value of the parameter $k \in \{3, 4, \ldots, 12\}$ that denotes how many instances the model selects to build the anytime performance profile that will predict anytime performance. We see that the best, i.e., minimal, MAE value is for $k = 11$. As such, we consider $k = 11$ to train the final model using all instances in the training data set.

3	4	5	6	7	8	9	10	11	12
1.799	1.745	1.709	1.686	1.672	1.660	1.644	1.648	**1.643**	1.645

Table 4.3: MAE given by LOOCV on the training problem instances. PLS, 2 objectives.

Figure 4.3 gives the distribution of the prediction errors for testing problem instances considering the final model trained with $k = 11$. In particular, the prediction error is considering in terms of the difference between the anytime measure of the predicted anytime performance profile for an instance in the

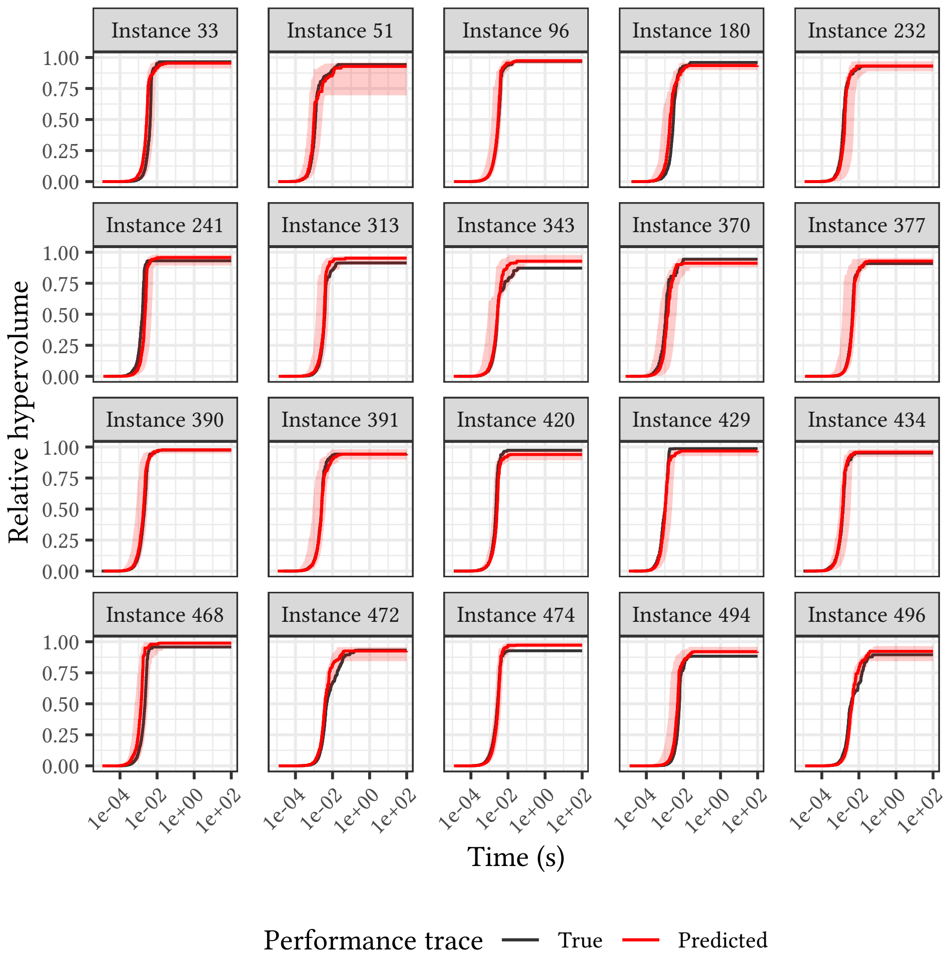


Figure 4.4: True and predicted anytime performance for 20 randomly selected testing problem instances. PLS, 2 objectives.

testing data set, and the anytime measure of the true anytime performance trace for that instance. We see that most error values are close to zero, and the median (red line) is also very close to zero. We do see one particularly large error, but this is not surprising given the fact that the density plots shown earlier in this section revealed that there were outliers in the testing data set. Overall, we consider that these results indicate that our empirical model can accurately predict the anytime measure of the PLS algorithm for previously unseen problem instances with $m = 2$ objectives.

Figure 4.4 shows the true and predicted anytime performance for 20 randomly selected testing problem instances. The black line corresponds to the true anytime trace gathered for that problem instance, the red region gives the range of anytime performance traces of the k problem instances selected to build the anytime performance profile used for the prediction, and the red line gives the median anytime trace of that predicted anytime performance

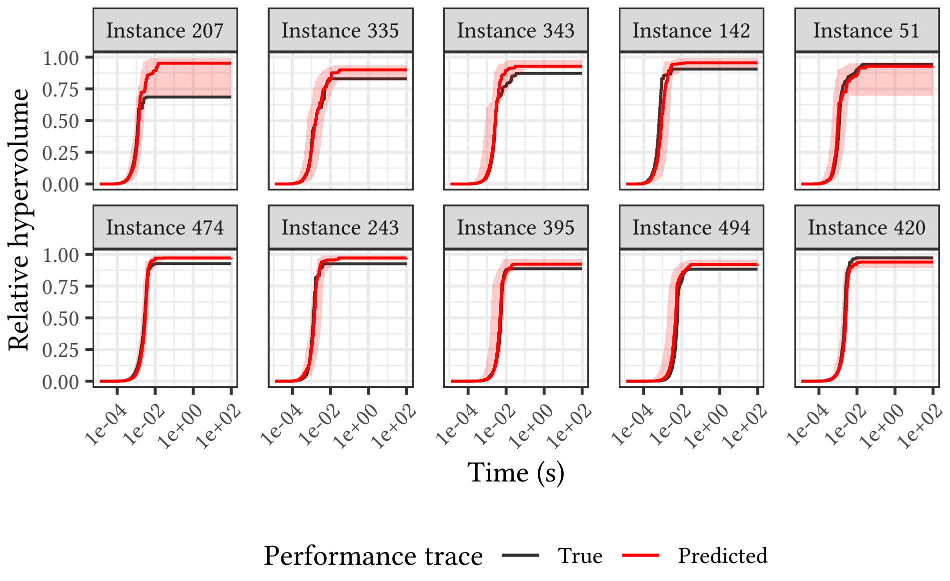


Figure 4.5: True and predicted anytime performance for testing problem instances that resulted in the largest prediction errors. PLS, 2 objectives.

profile. We observe that the true anytime trace is often within the red region and close to the median anytime trace. Moreover, the red region is often quite small. These observations indicate that our model is often selecting problem instances with similar anytime behavior for building the predicted anytime performance profile. Therefore, it appears that our model can accurately predict the anytime performance of the PLS algorithm for previously unseen instances with $m = 2$ objectives quite well. The true anytime trace and predicted anytime performance of all 100 testing problem instances is presented in Appendix A.1.

Figure 4.5 and Table 4.4 give the predicted and true anytime performance profiles and measures for the 10 testing problem instances that had the largest prediction errors. We see that the top 3 largest prediction errors are for instances with a small true anytime measure as we had anticipated in the beginning of this section. Note that, due to the stochastic nature of the algorithm, and because we only gathered one run for each instance, these large errors might be a consequence of lucky, or unlucky, runs of the algorithm on particular instances. As such, one aspect to be studied in the future is the impact that collecting multiple runs per instance has. Finally, we see that for the last of these instances, the predicted performance profile is already quite close to the true anytime behavior. As such, we consider that our model can accurately predict the anytime performance of the PLS algorithm for problem instances with $m = 2$ objectives.

Instance	Predicted measure	True measure	Error
207	90.13	68.53	21.60
335	89.34	82.97	6.37
343	93.47	87.21	6.26
142	95.56	90.57	4.99
51	89.35	94.29	-4.94
474	97.20	92.74	4.47
243	97.02	92.63	4.38
395	92.72	88.74	3.99
494	92.27	88.32	3.96
420	93.61	97.36	-3.74

Table 4.4: True and predicted anytime measure values for testing problem instances that resulted in the largest prediction errors. PLS, 2 objectives.

PLS — 3 Objectives

In this section, we analyze the results of predicting the anytime behavior of the PLS algorithm for problem instances with 3 objectives.

Figure 4.6 gives the density of anytime measure values for training and testing problem instances. We see that the distribution is similar for both types of instances, and that most instances have an anytime measure value greater than 90. Due to the small number of instances on the left tail, we expect that these might lead to erroneous predictions. We note that these results are similar to those of the previous section where we considered instances with 2 objectives. However, there are less significant outliers on the left tail.

Figure 4.7 gives the relation between the anytime measure and feature values for instances in the training data set , and Table 4.5 gives the corresponding Spearman's correlation coefficient between the features and anytime measure values. We see that the problem size feature, $\mathcal{F}_1$, has the most meaningful impact on anytime measure. We also see that features $\mathcal{F}_3$ and $\mathcal{F}_4$ appear to have a relevant impact for some instances, but that there are many instances that always have an anytime measure value of 100 regardless of those features. This

$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
0.559	0.208	-0.134	0.095

Table 4.5: Spearman's correlation coefficient between the features and anytime measures of the training instances. PLS, 3 objectives.

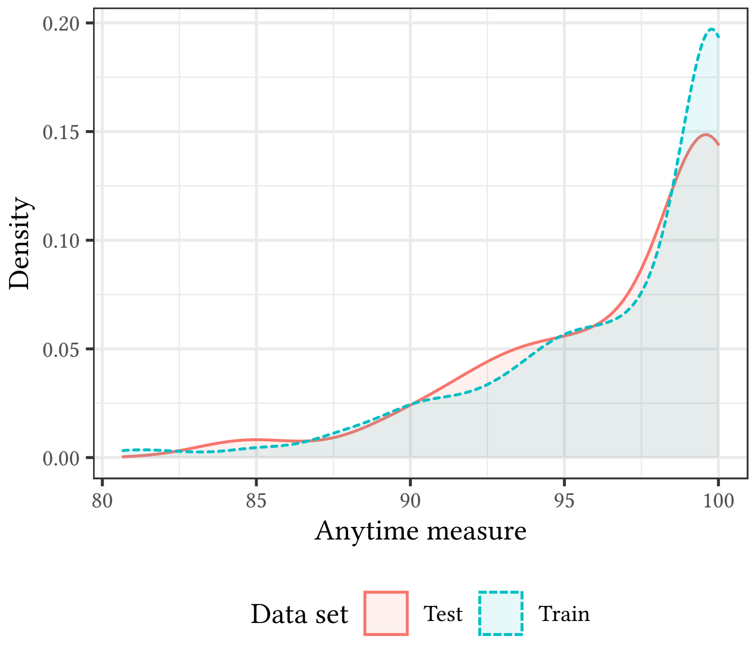


Figure 4.6: Anytime measure density plot for training and testing problem instances. PLS, 3 objectives.

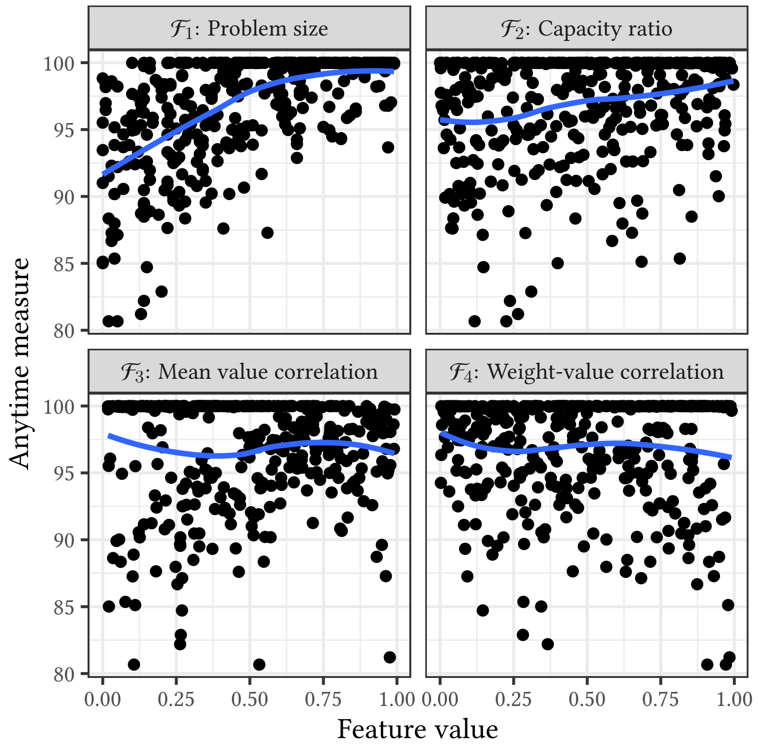


Figure 4.7: Relation between anytime measure and feature values for training problem instances. PLS, 3 objectives.

ξ_1	ξ_2	ξ_3	ξ_4
9.373	2.301	1.707	1.461

Table 4.6: Weights for the distance metric. PLS, 3 objectives.

happens because for some instances, especially for small values of n, the BHV-DP algorithm can find the efficient set. Therefore, in those cases, the anytime measure value for the PLS algorithm will be relative to the hypervolume of the non-dominated set. However, for other instances, the BHV-DP cannot find the efficient set within the allotted time, and the best known approximation, in terms of hypervolume, might be given by the PLS algorithm. Therefore, in those case, the anytime measure value for the PLS algorithm will be relative to its own approximation, and will have a value of 1. As a result, it seems that $\mathcal{F}_1$ is the feature that best indicates whether or not the PLS algorithm found the best known approximation, and the correlation features, $\mathcal{F}_3$ and $\mathcal{F}_4$, better explain the anytime measure for instances where it did not. Note that, in the previous case with 2 objectives, the BHV-DP algorithm always finds the efficient set within the allotted time, so we did not see this behavior. Table 4.6, gives the weights obtained from the corresponding correlation coefficients according to Equation (4.16).

Table 4.7 gives the MAE values obtained with the LOOCV method for each value of the parameter k considered. Since the minimum MAE value was found for $k = 9$, we will consider that value to train our final model on the complete training data set.

Figure 4.8 shows the distribution of the prediction errors in terms of anytime measure value for testing problem instances using the final model. We see that the error for most instances, as well as the median error, are close to zero. Still, it seems that there are slightly more errors with negative values, which means that the model is under-estimating the true anytime measure. This is in contrast to the expected result that the largest errors would be related to the outliers in the left tail, which is what we saw in the previous section for instances with 2 objectives. One reason for this, might be related to what was discussed previously that for some instances the best known approximation

3	4	5	6	7	8	9	10	11	12
1.801	1.820	1.830	1.816	1.821	1.792	**1.784**	1.811	1.840	1.866

Table 4.7: MAE given by LOOCV on the training problem instances. PLS, 3 objectives.

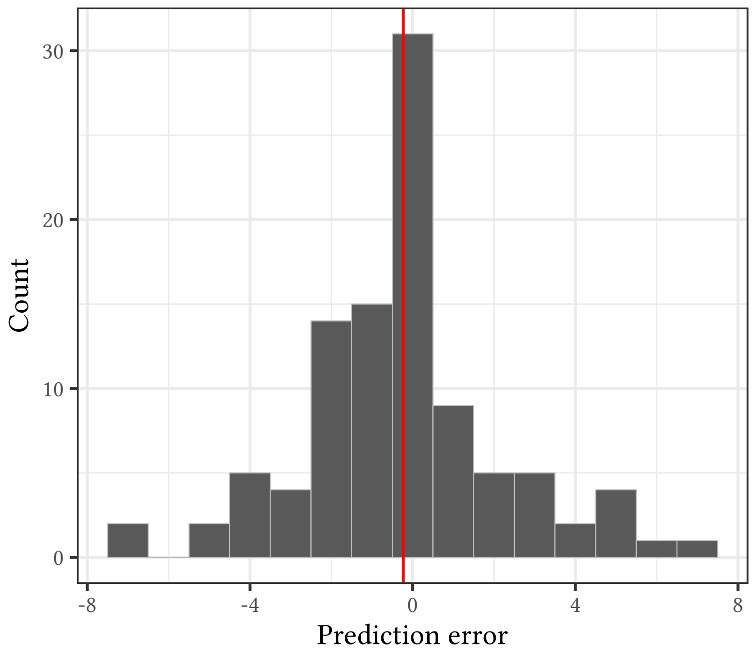


Figure 4.8: Histogram of the prediction error in terms of anytime measure for testing problem instances. The red line corresponds to the median. PLS, 3 objectives.

is given by the BHV-DP algorithm, whereas for others it is given by the PLS algorithm. As a results of this, for similar instances, it might happen that the BHV-DP was able to find the efficient set within the allotted time, but for others it did not. As such, we might have similar instances with distinct relative anytime measure values, i.e., close to 100 if PLS found the best known solution, and significantly below 100 if BHV-DP found the optimal solution. Despite this, the results show that the model can accurately predict the anytime measure obtained by the PLS algorithm for instances with 3 objectives in most cases.

Figure 4.9 gives the true and predicted anytime performance for 20 randomly selected testing problem instances. We see that the results are generally quite good, and that the model can accurately predict the anytime performance for previously unseen instances. Nonetheless, there are some notable outliers, e.g., instances 148, 165, 193, 252, and 305. However, Table 4.8 shows that, with the exception of instance 193, these outliers are in the list of the 10 worst predictions. Thus, we were a bit unlucky with the random sample, and looking at the full results in Appendix A.2, it is clear that the model can very often predict the anytime performance of the PLS algorithm for problem instances with 3 objectives.

Figure 4.10 and Table 4.8 show the predicted and true anytime performance and anytime measure for the 10 testing problem instances with the worst pre-

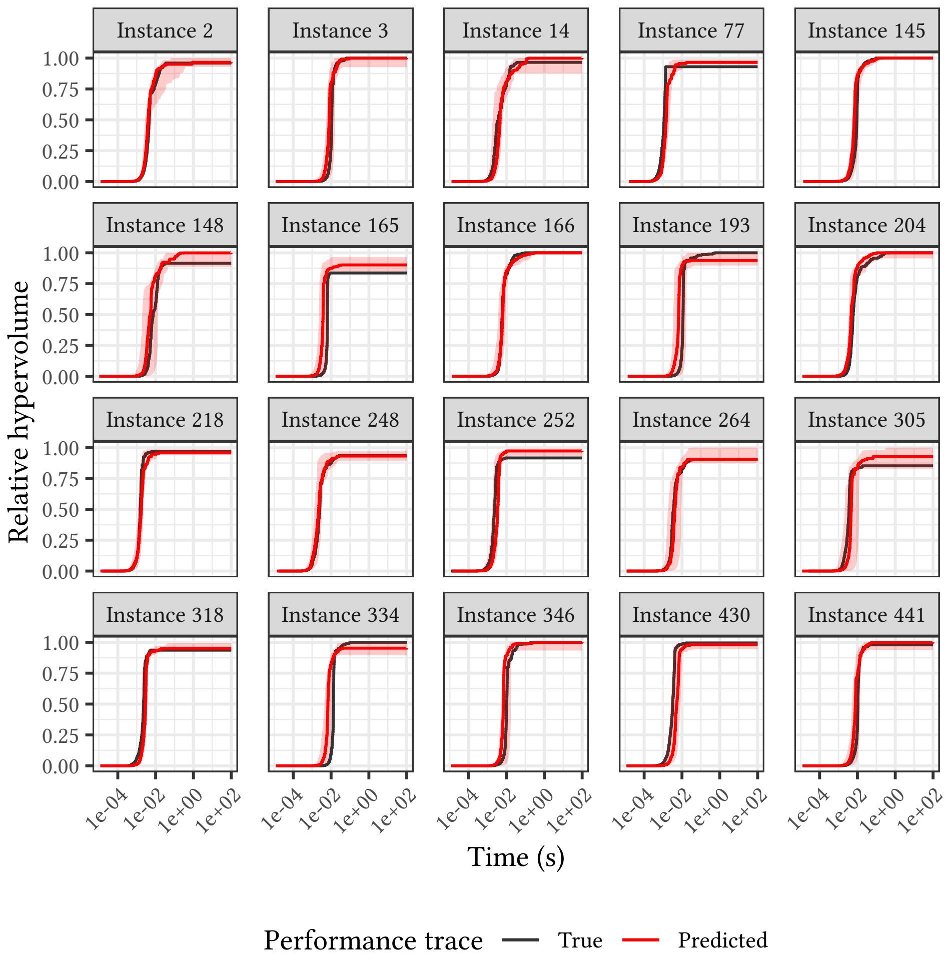


Figure 4.9: True and predicted anytime performance for 20 randomly selected testing problem instances. PLS, 3 objectives.

diction error. Some of the largest errors are for instances with a true anytime measure very close to 100, which is consistent with the analysis regarding instances whose best known solution may be found by the PLS or BHV-DP algorithm. There are also some errors for instances with a small true anytime measure, which is to be expected due to the small number of instances with those anytime measure values. Nonetheless, we see that the overall anytime behavior for these instances is not very far from the true anytime trace.

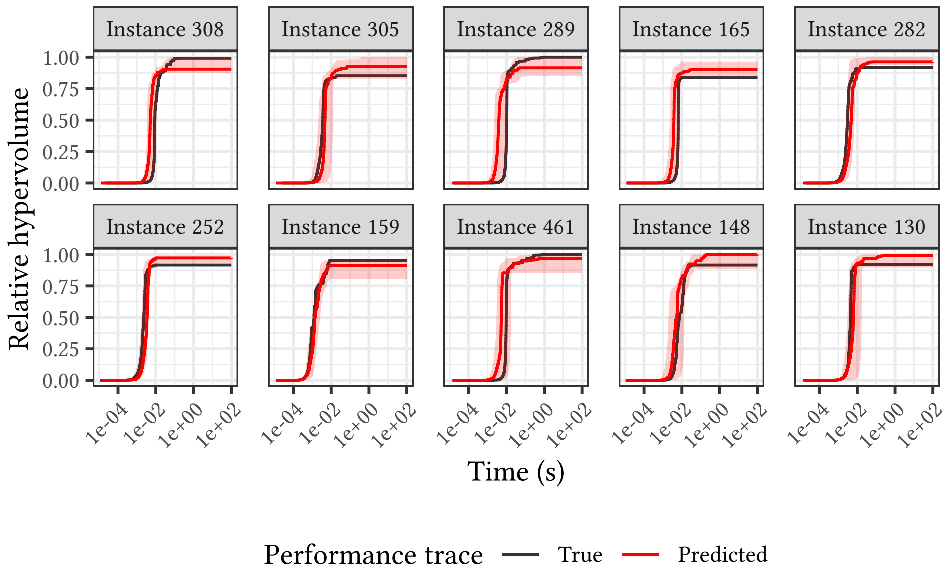


Figure 4.10: True and predicted anytime performance for testing problem instances that resulted in the largest prediction errors. PLS, 3 objectives.

Instance	Predicted measure	True measure	Error
308	92.04	99.16	-7.12
305	92.23	85.22	7.01
289	93.02	99.98	-6.96
165	90.14	83.65	6.48
282	97.17	91.75	5.42
252	96.72	91.59	5.13
159	90.16	95.22	-5.06
461	94.93	99.98	-5.05
148	96.26	91.56	4.70
130	96.86	92.19	4.67

Table 4.8: True and predicted anytime measure values for testing problem instances that resulted in the largest prediction errors. PLS, 3 objectives.

PLS — 5 Objectives

In this section, we analyze the results of predicting the anytime behavior of the PLS algorithm for problem instances with 5 objectives.

Figure 4.11 shows the density of the anytime measure for testing and training problem instances. We see a similar density for the two data sets. However, the testing data set seems to have more instances with an anytime measure around 90, and less instances with an anytime measure close to 100. As such, we expect to be able to generally predict anytime performance, but that our largest errors might be for instances with an anytime measure close to 90. Compared to the two previous cases where we considered instances with 2 and 3 objectives, the anytime measure values are much closer to 100, i.e., the maximum possible value. This is a consequence of the best known solution for instances with 5 objectives being very often found by the PLS algorithm, instead of the BHV-DP algorithm, since the latter is not often able to find the efficient set in the given amount of time.

Figure 4.12 gives the relation between the features values and anytime measure for instances in the training data set, and Table 4.9 gives the corresponding Spearman's correlation coefficient. We see that, unlike in the previous cases, all features have a give a small correlation coefficient, and as such the weights for the distance metric, shown in Table 4.10, are not too different from each other.

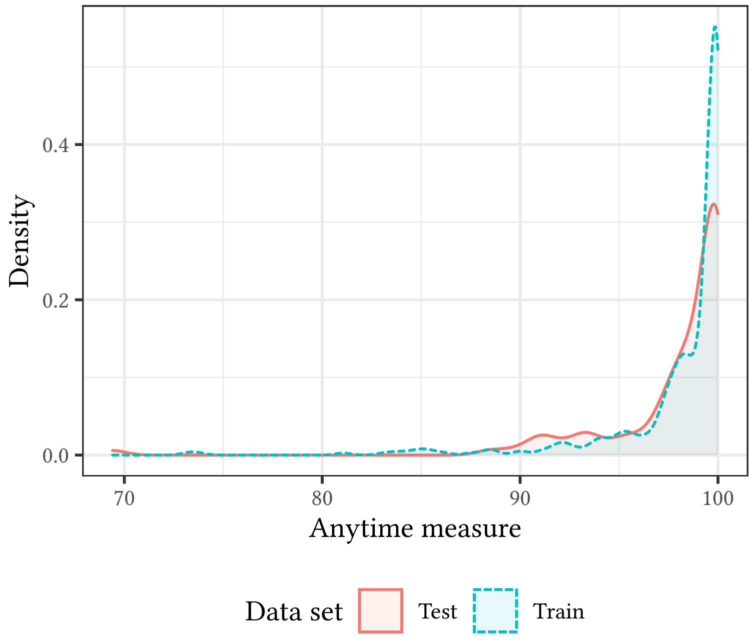


Figure 4.11: Anytime measure density plot for training and testing problem instances. PLS, 5 objectives.

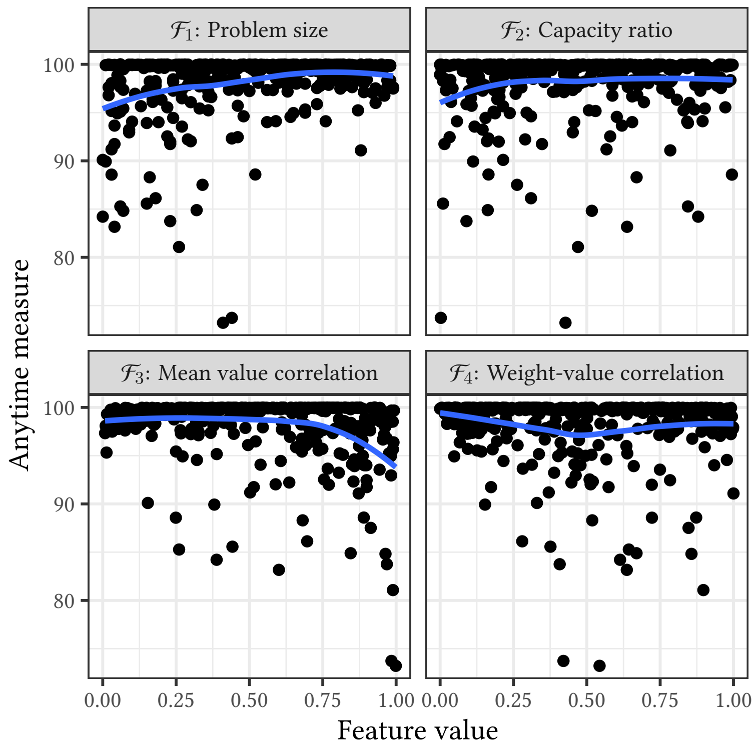


Figure 4.12: Relation between anytime measure and feature values for training problem instances. PLS, 5 objectives.

$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
0.130	0.115	-0.063	-0.092

Table 4.9: Spearman's correlation coefficient between the features and anytime measures of the training instances. PLS, 5 objectives.

ξ_1	ξ_2	ξ_3	ξ_4
1.680	1.583	1.286	1.445

Table 4.10: Weights for the distance metric. PLS, 5 objectives.

Table 4.11 gives the MAE values obtained by LOOCV on the training problem instances for values of parameter $k = 3, 4, \ldots, 12$. The best result was found for $k = 5$, which we use to build the final model.

Figure 4.13 gives the histogram of prediction errors for the testing problem instances. We see that most errors, and the median, are close to zero, and that, as expected, the most significant errors are positive, i.e., a consequence of over-estimating the anytime performance. Nonetheless, we see that the model can accurately predict the anytime measure values very often.

Figure 4.14 shows the true and predicted anytime performance of 20 randomly selected testing problem instances. We see that for most instances the prediction is quite good, but there are some outliers. Most notably, instance 412 has a very significant error, but it is worth noting that it is in the fact the testing instance that has the largest error and a clear outlier. Compared to the previous scenarios with 2 and 3 objectives, we seem to have larger red regions,

3	4	5	6	7	8	9	10	11	12
1.530	1.525	**1.475**	1.521	1.546	1.555	1.559	1.539	1.534	1.536

Table 4.11: MAE given by LOOCV on the training problem instances. PLS, 5 objectives.

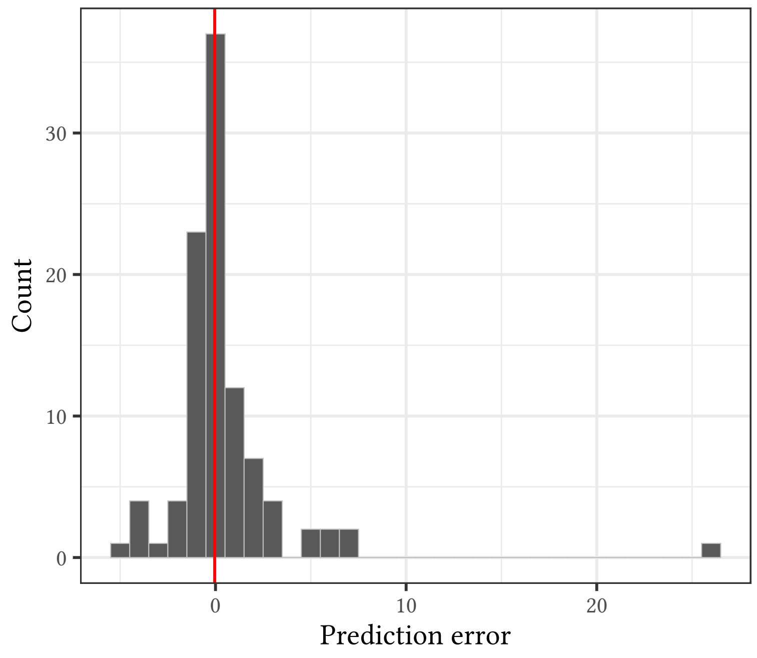


Figure 4.13: Histogram of the prediction error in terms of anytime measure for testing problem instances. The red line corresponds to the median. PLS, 5 objectives.

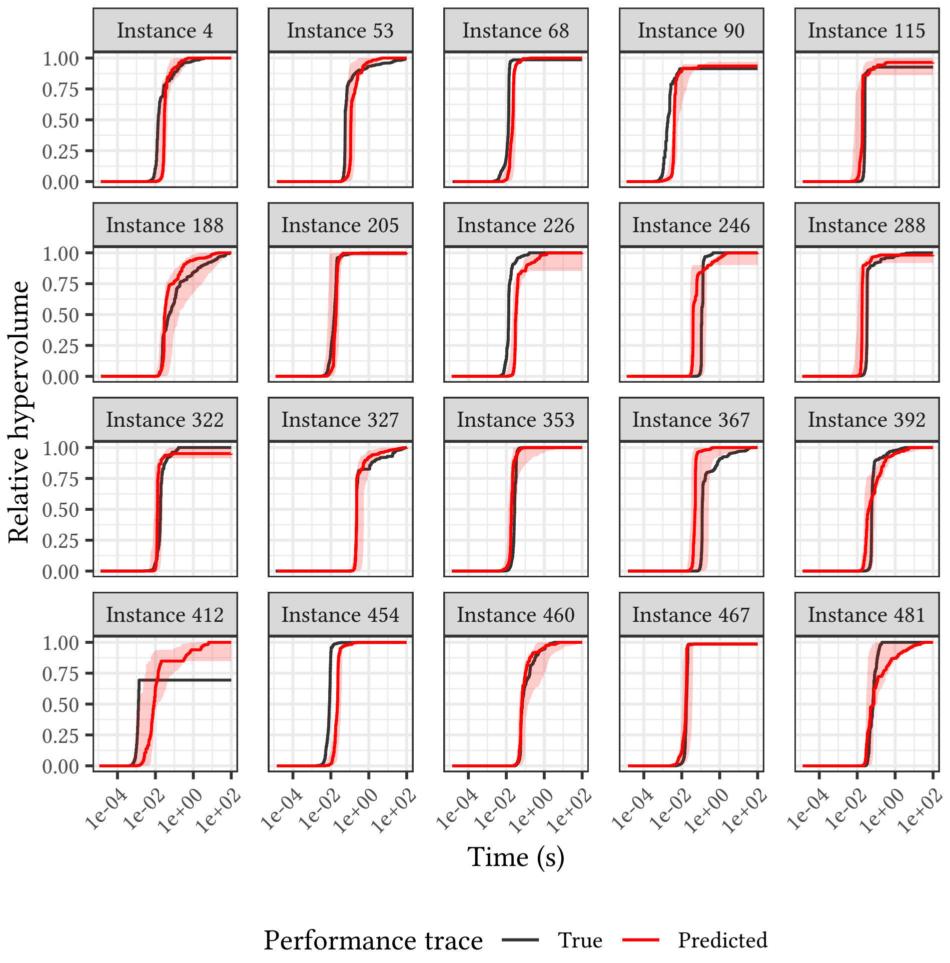


Figure 4.14: True and predicted anytime performance for 20 randomly selected testing problem instances. PLS, 5 objectives.

which corresponds to the region of anytime traces of the training instances selected for prediction. This suggests that it is harder to find instances with similar anytime behavior for a larger number of objectives, which is not too surprising. The results for every testing problem instance can be found in Appendix A.3.

Figure 4.15 and Table 4.12 give the true and predicted anytime measure and performance for the testing problem instances with the largest prediction error. As expected, most of the largest errors are for instances with a true measure close to 90. Nonetheless, we see that the model can already predict the anytime performance accurately for the last 5 of these instances, and also taking into account the full results, we see that our model can accurately predict anytime behavior often.

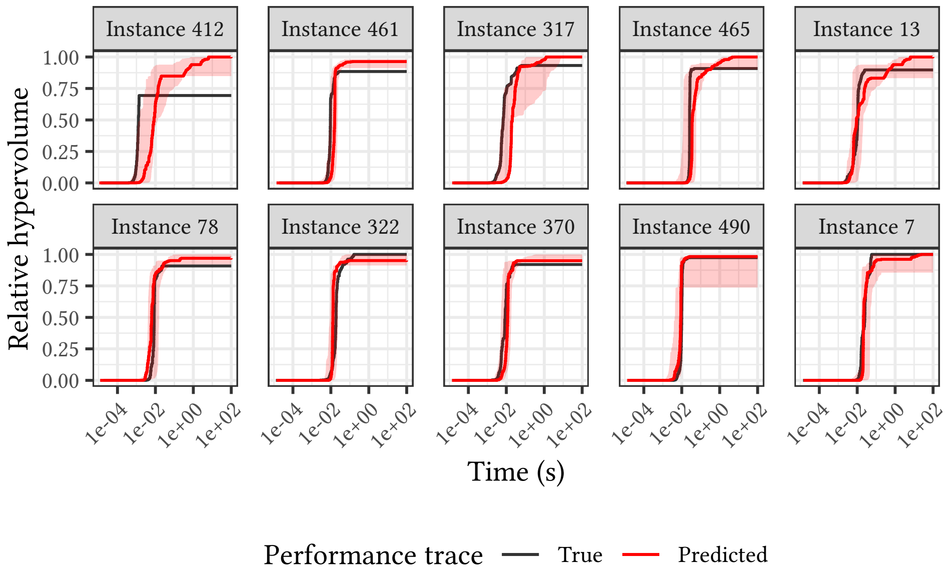


Figure 4.15: True and predicted anytime performance for testing problem instances that resulted in the largest prediction errors. PLS, 5 objectives.

Instance	Predicted measure	True measure	Error
412	95.64	69.40	26.23
461	95.42	88.46	6.95
317	99.75	93.25	6.50
465	96.96	90.79	6.16
13	95.29	89.71	5.59
78	96.28	90.82	5.46
322	95.25	99.97	-4.73
370	96.53	91.98	4.55
490	93.24	97.43	-4.18
7	96.19	99.97	-3.79

Table 4.12: True and predicted anytime measure values for testing problem instances that resulted in the largest prediction errors. PLS, 5 objectives.

BHV-DP — 2 Objectives

In this section, we analyze the prediction of anytime performance of the BHV-DP algorithm for problem instances with 2 objectives.

Figure 4.16 gives the density of anytime measure for problem instances in the testing training data sets. We can see that the anytime measure is very close to 100 for most instances. This is due to the fact that the BHV-DP algorithm can often find the efficient set quickly for the instances and time budget we considered. However, there are some testing instances with an anytime measure below 97.5 that we expect to lead to large errors since there are very little training instances with similar values.

Figure 4.17 shows the relation between the instance features and the anytime measure values, and Table 4.13 gives the Spearman's correlation coefficients. Instance features $\mathcal{F}_1$ and $\mathcal{F}_3$ seem to have the most significant impact on anytime measure values, followed by $\mathcal{F}_4$. On the other hand, $\mathcal{F}_2$ seems to have little or no impact. Table 4.14 gives the weights for the distance metric computed from the correlation values according to Equation (4.16).

Table 4.15 gives the MAE obtained by LOOCV for values of parameter $k = 3, 4, \ldots, 12$. The best value was found for $k = 9$, which consequently is the value that we will use to build the final model considering all problem instances in the training data set.

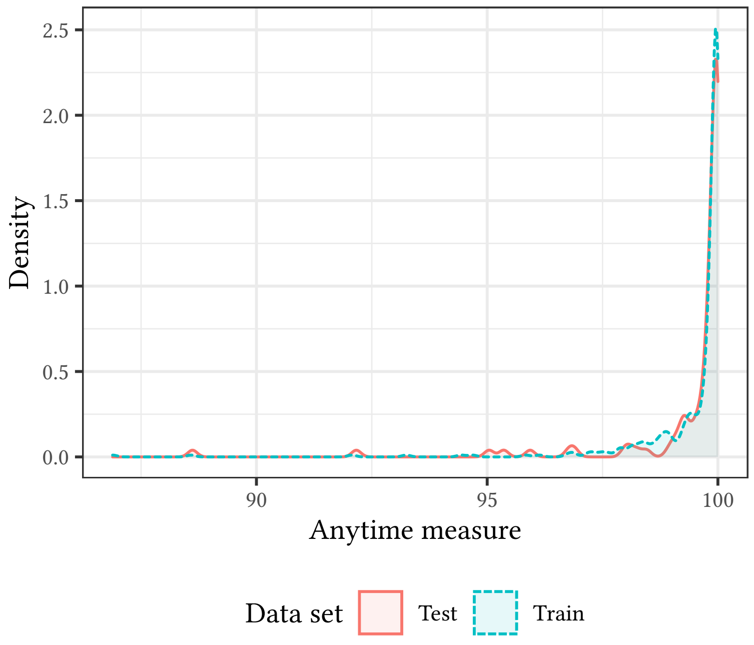


Figure 4.16: Anytime measure density plot for training and testing problem instances. BHV-DP, 2 objectives.

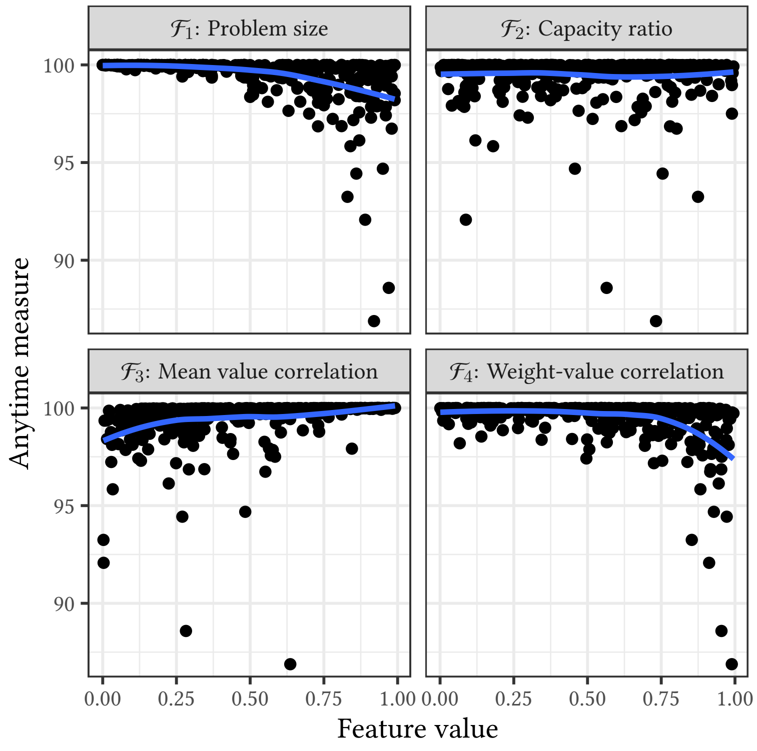


Figure 4.17: Relation between anytime measure and feature values for training problem instances. BHV-DP, 2 objectives.

$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
-0.576	-0.037	0.658	-0.356

Table 4.13: Spearman's correlation coefficient between the features and anytime measures of the training instances. BHV-DP, 2 objectives.

ξ_1	ξ_2	ξ_3	ξ_4
10.016	1.158	13.907	4.149

Table 4.14: Weights for the distance metric. BHV-DP, 2 objectives.

3	4	5	6	7	8	9	10	11	12
0.252	0.249	0.242	0.238	0.234	0.233	**0.226**	0.231	0.226	0.229

Table 4.15: MAE given by LOOCV on the training problem instances. BHV-DP, 2 objectives.

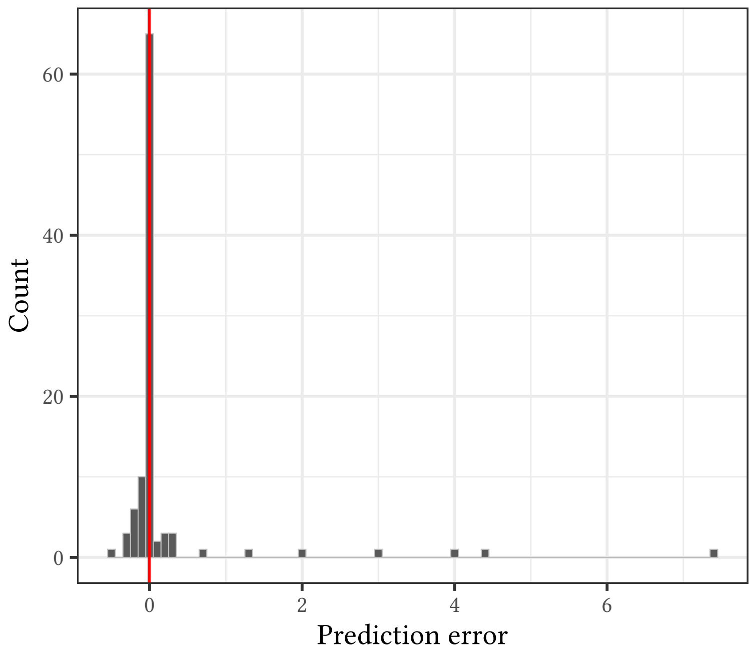


Figure 4.18: Histogram of the prediction error in terms of anytime measure for testing problem instances. The red line corresponds to the median. BHV-DP, 2 objectives.

Figure 4.18 shows the prediction errors, in terms of anytime measure, over all 100 testing problem instances. The errors are for the most part quite small, which is expected given that most instances have a similar anytime measure value as discussed earlier. We also see some significant positive errors that, as expected, correspond to instances that have a small anytime measure value.

Figure 4.19 gives the true and predicted anytime performance for 20 randomly selected testing problem instances. The results for all 100 testing problem instances can be found in Appendix A.4. We note that the red region of anytime traces used to predict the anytime performance of each instance is generally quite large compared what was seen before when predicting the anytime behavior of the PLS algorithm. This suggests that accurately predicting the anytime performance of the BHV-DP algorithm is more difficult, and we may need to consider more instances or instance features for the model to be more precise. Still, the tendency of anytime behavior is correctly predicted in most cases. As such, we consider that the model can accurately predict the anytime behavior of the BHV-DP algorithm for instances with 2 objectives.

Figure 4.20 and Table 4.16 show the true and predicted anytime measure and anytime performance for the testing problem instances with the largest prediction errors. We see that, as expected, the worst predictions are for instances that resulted in a small true anytime measure value.

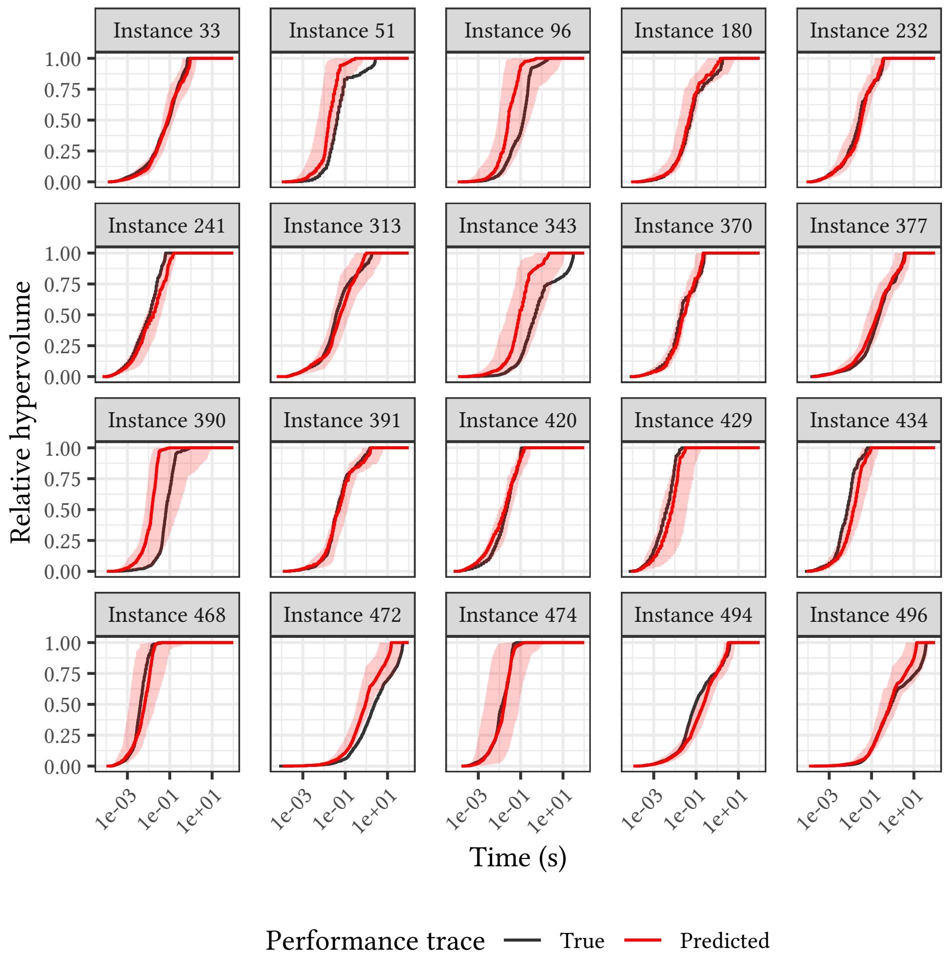


Figure 4.19: True and predicted anytime performance for 20 randomly selected testing problem instances. BHV-DP, 2 objectives.

Instance	Predicted measure	True measure	Error
472	96.06	88.61	7.45
343	99.42	95.04	4.37
496	96.19	92.16	4.03
159	98.89	95.93	2.96
424	98.87	96.90	1.97
467	96.66	95.36	1.30
30	98.71	97.98	0.73
346	99.21	99.68	-0.47
294	97.73	98.08	-0.35
440	99.40	99.06	0.34

Table 4.16: True and predicted anytime measure values for testing problem instances that resulted in the largest prediction errors. BHV-DP, 2 objectives.

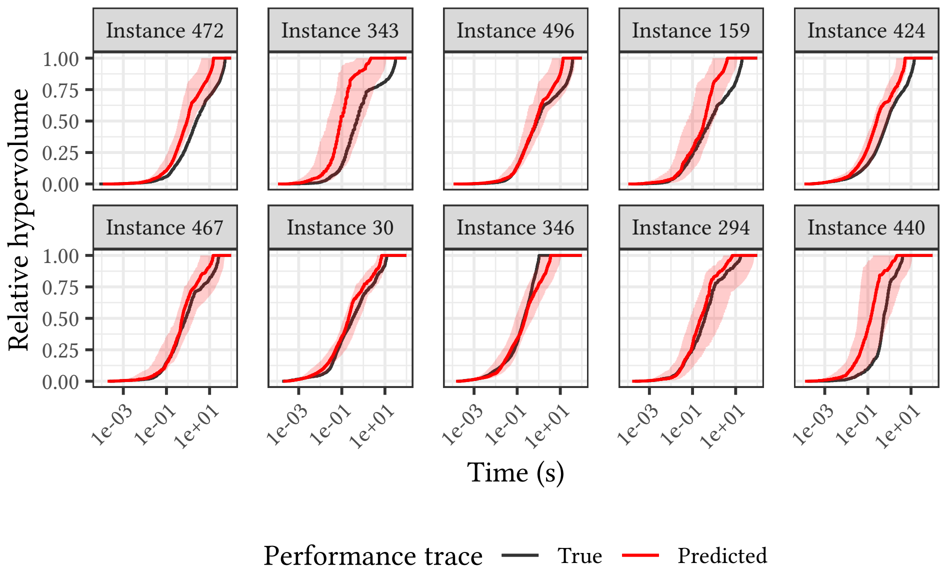


Figure 4.20: True and predicted anytime performance for testing problem instances that resulted in the largest prediction errors. BHV-DP, 2 objectives.

BHV-DP — 3 Objectives

In this section we analyze the prediction of anytime performance of the BHV-DP algorithm for problem instances with 3 objectives. Figure 4.21 gives the distribution of anytime measure values for the training and testing problem instances. We see a similar distribution between both data sets, but a large range of anytime measure values. This large range of values is explained by the fact that the BHV-DP can sometimes take a long time to find a good approximation on the considered instances. Given that the number of instances in the training data set is not very extensive when taking into account the combination of parameters, having a large range of anytime measure values may suggest that there are not many instances with similar anytime behavior and instance features, which could result in degraded prediction quality.

Figure 4.22 gives the relation between the feature values and anytime measure values for the training problem instances, and Table 4.17 gives the corresponding Spearman's correlation coefficient. The results indicate that all

$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
-0.479	-0.247	0.684	-0.313

Table 4.17: Spearman's correlation coefficient between the features and anytime measures of the training instances. BHV-DP, 3 objectives.

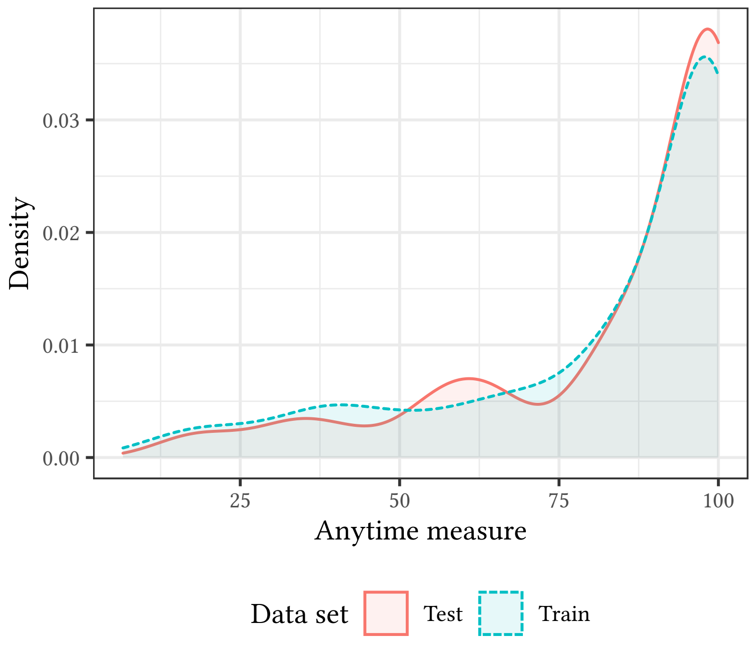


Figure 4.21: Anytime measure density plot for training and testing problem instances. BHV-DP, 3 objectives.

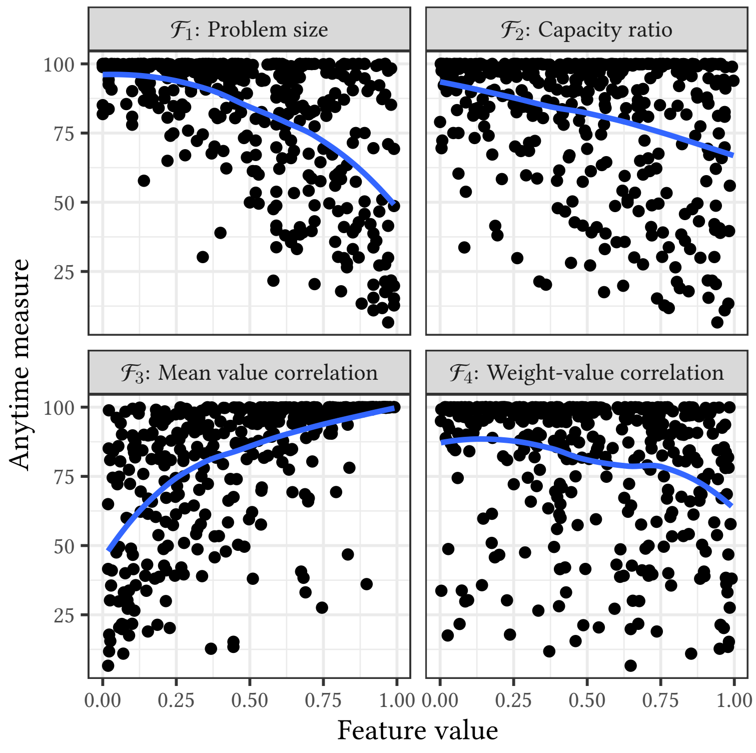


Figure 4.22: Relation between anytime measure and feature values for training problem instances. BHV-DP, 3 objectives.

ξ_1	ξ_2	ξ_3	ξ_4
6.782	2.684	15.411	3.494

Table 4.18: Weights for the distance metric. BHV-DP, 3 objectives.

3	4	5	6	7	8	9	10	11	12
4.908	5.113	5.088	5.173	5.193	5.233	5.298	5.333	5.442	5.467

Table 4.19: MAE given by LOOCV on the training problem instances. BHV-DP, 3 objectives.

instance features have some impact on the anytime measure value, with the most significant being $\mathcal{F}_3$ and $\mathcal{F}_1$. Table 4.18 gives the weight values for the distance metric obtained with Equation (4.16) from the correlation coefficients.

Table 4.19 gives the MAE values obtained by LOOCV for $k = 3, 4, \ldots, 12$. The best value was found for $k = 3$, which we use to build the final model.

Figure 4.23 gives the histogram of prediction errors for the testing problem instances. We can see that most errors are close to zero. However, we see a considerable number of instances with significant errors, which is not unexpected given the large range of anytime measure values discussed before. We expect

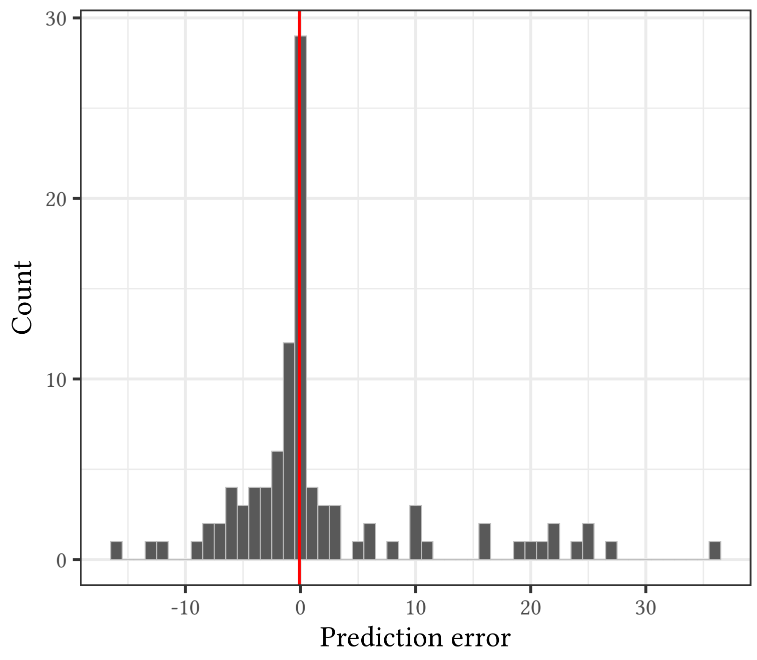


Figure 4.23: Histogram of the prediction error in terms of anytime measure for testing problem instances. The red line corresponds to the median. BHV-DP, 3 objectives.

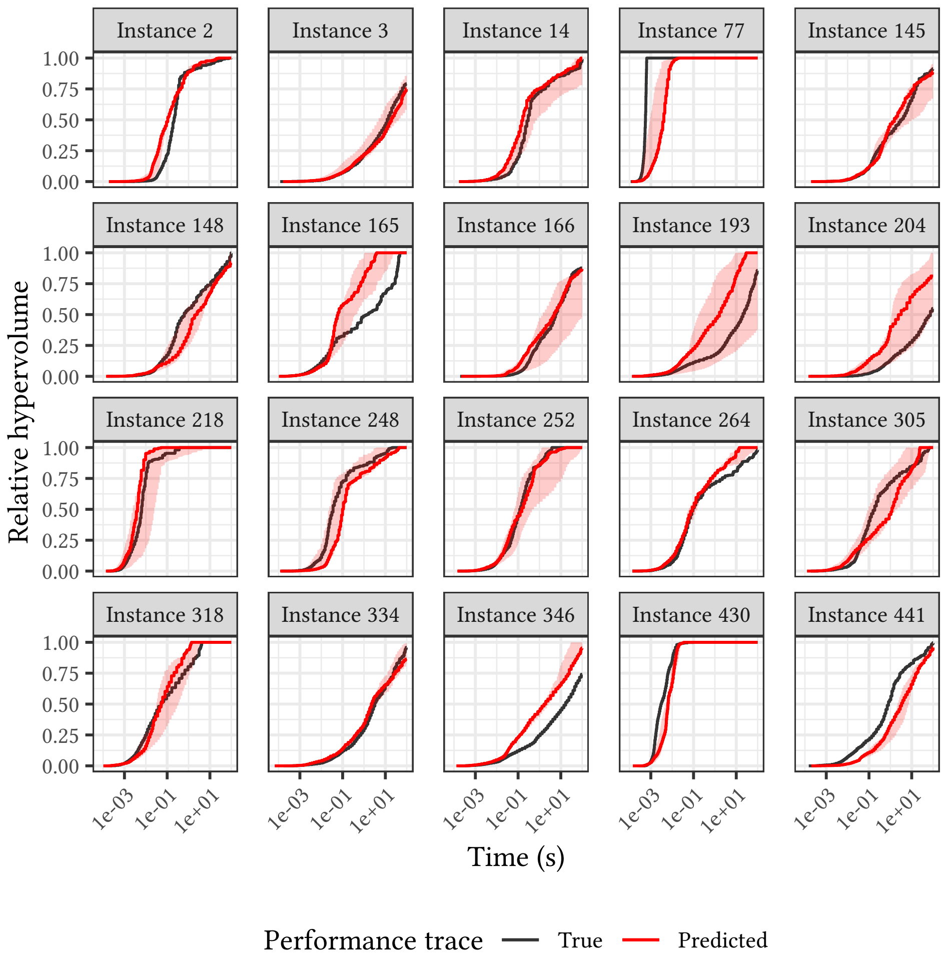


Figure 4.24: True and predicted anytime performance for 20 randomly selected testing problem instances. BHV-DP, 3 objectives.

that by considering a larger number of training instances the results from our model could be significantly improved. Nonetheless, as it stands, the quality of prediction in terms of anytime measure is not very good for this case.

Figure 4.24 gives the true and predicted anytime performance for 20 randomly selected testing instances. The results for every testing problem instance can be found in Appendix A.5. As in the previous case, we see that the region of anytime traces, i.e., the red region, is often large despite the fact that we are selecting $k = 3$ instances. This indicates that the model fails to find instances that have similar anytime performance, which, as discussed earlier, is expected due to the large range of anytime measure values. Moreover, we see that in quite a few cases the prediction is not very good. However, on a more positive note, we see that the overall tendency of anytime behavior can often be predicted. This suggests that by considering more instances the model could better predict anytime behavior.

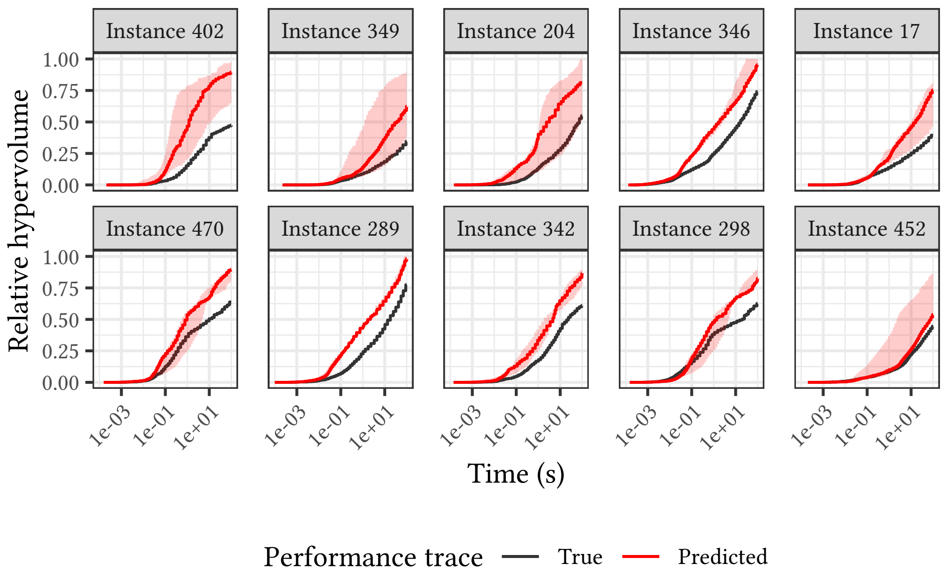


Figure 4.25: True and predicted anytime performance for testing problem instances that resulted in the largest prediction errors. BHV-DP, 3 objectives.

Instance	Predicted measure	True measure	Error
402	78.92	42.91	36.01
349	54.12	26.67	27.45
204	68.01	42.91	25.11
346	85.88	61.20	24.68
17	56.65	32.91	23.74
470	79.33	57.35	21.99
289	83.33	61.72	21.61
342	74.42	53.32	21.09
298	75.30	55.63	19.67
452	53.31	34.62	18.69

Table 4.20: True and predicted anytime measure values for testing problem instances that resulted in the largest prediction errors. BHV-DP, 3 objectives.

Figure 4.25 and Table 4.20 give the true and predicted anytime measure and performance for the testing problem instances with the largest prediction errors. We see that often, the true anytime trace lies on the boundary of the red region. Moreover, we see that the largest errors happen for instances whose true anytime measure is not very large, despite there being more instances with large anytime measure values as seen earlier in the density plot. This reinforces the idea that by considering more instances the quality of the prediction could be significantly improved.

BHV-DP — 5 Objectives

In this section, we analyze the prediction of anytime performance of the BHV-DP algorithm for problem instances with 5 objectives. Figure 4.26 gives the distribution of anytime measure values for the training and testing problem instances. We see a similar distribution between both data sets, and that the generated instances cover the whole range of possible anytime measure values. As in the previous case, given the fact that we do not have a very extensive set of training instances, this is likely to result in large prediction errors.

Figure 4.27 gives the relation between the instance features and anytime measure on training problem instances, and Table 4.21 gives the corresponding Spearman's correlation coefficient. We see that all features have some

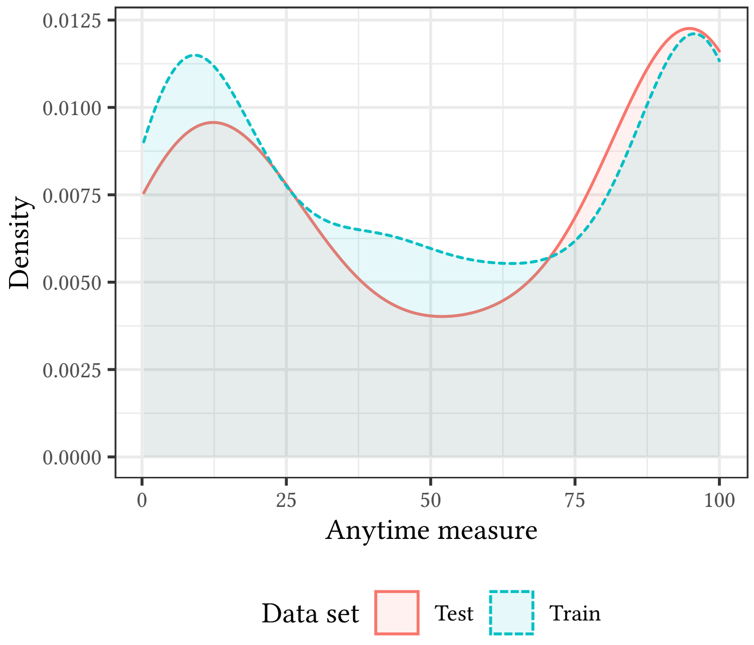


Figure 4.26: Anytime measure density plot for training and testing problem instances. BHV-DP, 5 objectives.

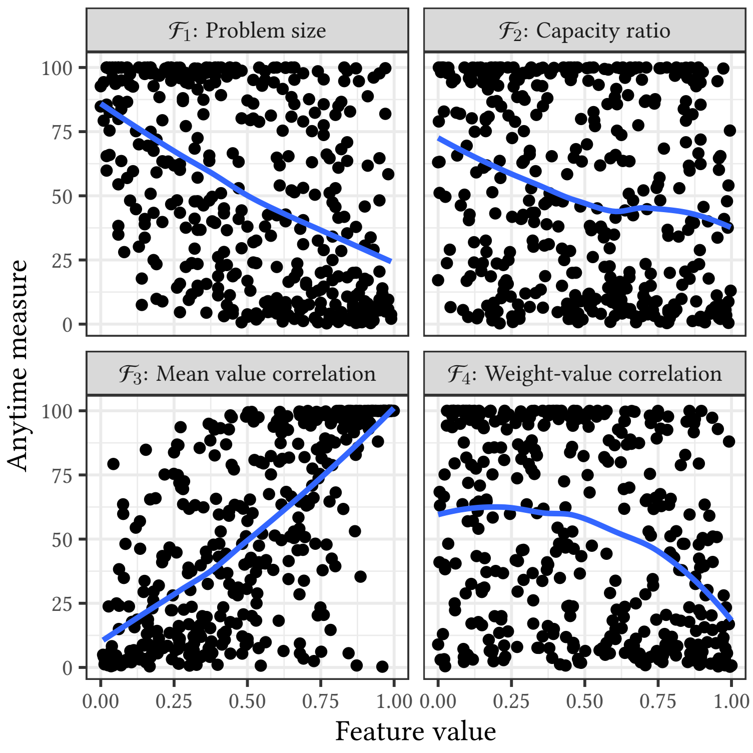


Figure 4.27: Relation between anytime measure and feature values for training problem instances. BHV-DP, 5 objectives.

$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
-0.489	-0.233	0.719	-0.322

Table 4.21: Spearman's correlation coefficient between the features and anytime measures of the training instances. BHV-DP, 5 objectives.

ξ_1	ξ_2	ξ_3	ξ_4
7.061	2.542	17.735	3.630

Table 4.22: Weights for the distance metric. BHV-DP, 5 objectives.

impact on anytime measure, but features $\mathcal{F}_3$ and $\mathcal{F}_1$ are the ones with the most significant impact. Table 4.22 gives the weights calculated with Equation (4.16).

Table 4.23 gives the MAE obtained by LOOCV for $k = 3, 4, \ldots, 12$. The best value was found for $k = 4$, which we use to build our final model.

Figure 4.28 gives the prediction errors for the testing problem instances. We see that the median and largest concentration of errors is close to zero. However, there are many large prediction errors. This is not unexpected, since a larger number of objectives will give rise to more distinct behaviors. There-

3	4	5	6	7	8	9	10	11	12
8.686	**8.636**	8.859	8.794	8.768	9.022	8.963	8.979	8.917	9.105

Table 4.23: MAE given by LOOCV on the training problem instances. BHV-DP, 5 objectives.

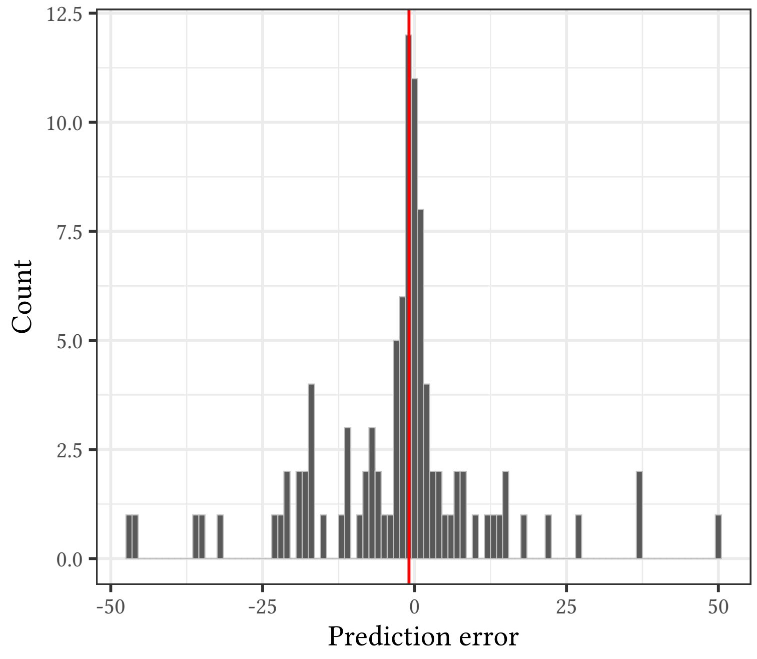


Figure 4.28: Histogram of the prediction error in terms of anytime measure for testing problem instances. The red line corresponds to the median. BHV-DP, 5 objectives.

fore, we consider that more instances or features are needed to precisely predict anytime measure for the BHV-DP algorithm on instances with 5 objectives.

Figure 4.29 gives the true and predicted anytime performance of 20 randomly selected testing problem instances. The results for all 100 testing problem instances are shown in Appendix A.6. We see that the red region of anytime traces that make up the predicted anytime performance profile is often quite large, and in a few cases the true tendency of anytime performance deviates significantly from the predicted anytime performance. As such, we consider that the overall prediction is not very good and could be further improved.

Figure 4.30 and Table 4.24 show the true and predicted anytime measure and performance profile for testing problem instances with the largest prediction errors. It is quite clear that the model is selecting instances that have quite different anytime behavior, so more information is needed to accurately predict anytime performance.

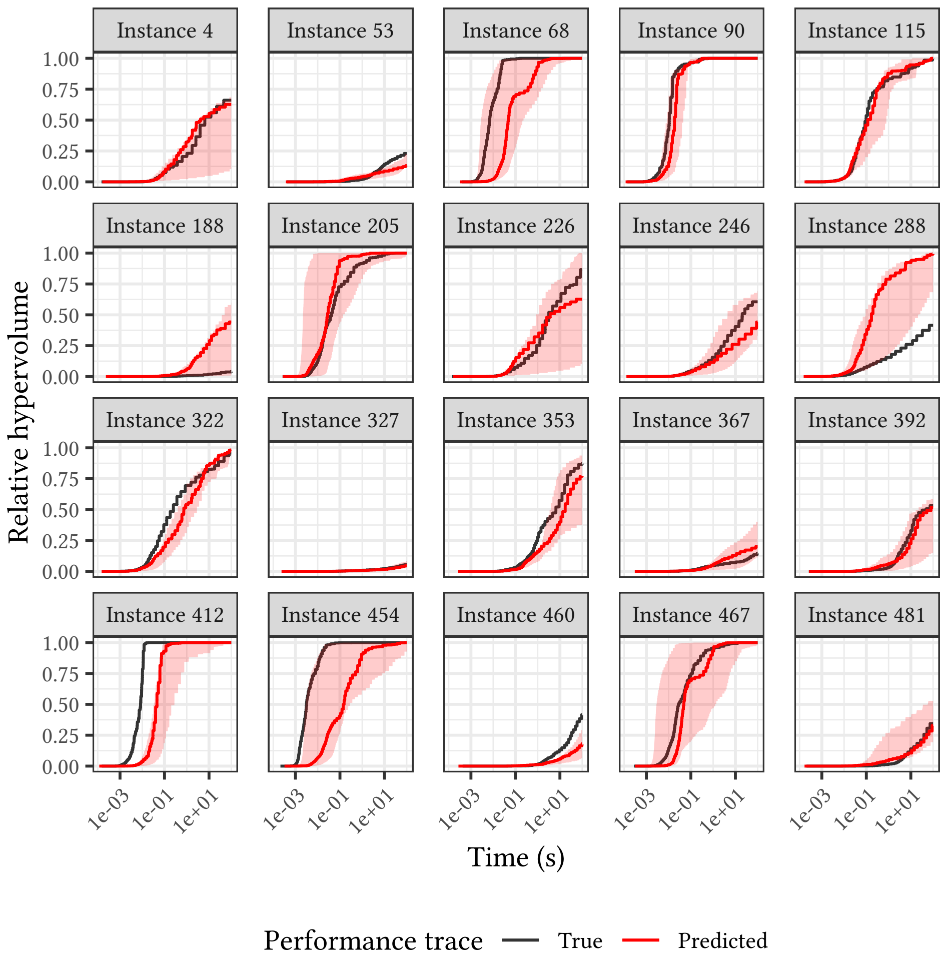


Figure 4.29: True and predicted anytime performance for 20 randomly selected testing problem instances. BHV-DP, 5 objectives.

Instance	Predicted measure	True measure	Error
288	85.92	35.85	50.07
317	51.82	98.50	-46.67
102	48.17	94.18	-46.01
400	56.81	19.42	37.39
70	62.19	24.99	37.20
465	61.92	98.17	-36.25
13	58.40	93.52	-35.12
217	52.65	84.51	-31.86
188	29.85	3.05	26.80
226	51.14	74.61	-23.47

Table 4.24: True and predicted anytime measure values for testing problem instances that resulted in the largest prediction errors. BHV-DP, 5 objectives.

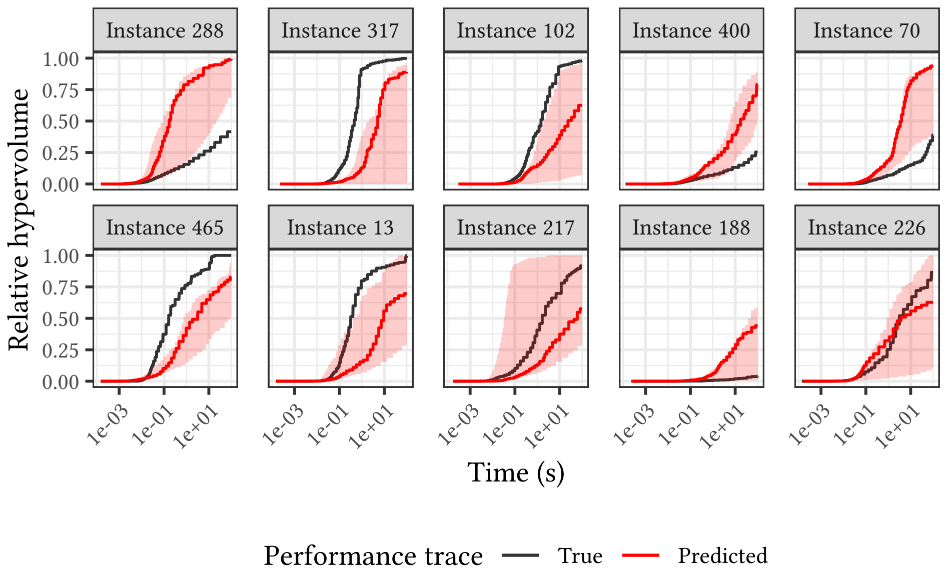


Figure 4.30: True and predicted anytime performance for testing problem instances that resulted in the largest prediction errors. BHV-DP, 5 objectives.

GEPS — 2 Objectives

In this section, we conclude our experimental analysis by looking at the prediction of anytime performance for the GEPS algorithm on problem instances with 2 objectives.

Figure 4.31 gives the density plot of anytime measure for training and testing problem instances. We can see that virtually all instances have an anytime measure close to 100. For this reason, we show in Figure 4.32, the density of instances with an anytime measure value greater than 99.9. We can see that the density is similar for both data sets. Still, we see a small number of testing problem instances below 99.9, but no such instances in the training data set. As such, we expect our model to give large prediction errors for these.

Figure 4.33 shows the relation between the instance features and anytime measure for the training problem instances, and Figure 4.34 shows the same plot for anytime measure values greater than 99.9. Moreover, Table 4.25 gives the Spearman's correlation coefficient, on all training instances. We see that instance feature $\mathcal{F}_3$ has the most significant impact on anytime measure. Then, features $\mathcal{F}_1$ and $\mathcal{F}_4$ also have some impact but to a lesser degree, and feature $\mathcal{F}_2$ has little impact. Table 4.26 gives the weights calculated with Equation (4.16) according to the correlation coefficients.

Table 4.27 shows the MAE values obtained by the LOOCV method for $k = 3, 4, \ldots, 12$. Thus, we choose $k = 7$ to build the final model has it has the best MAE value.

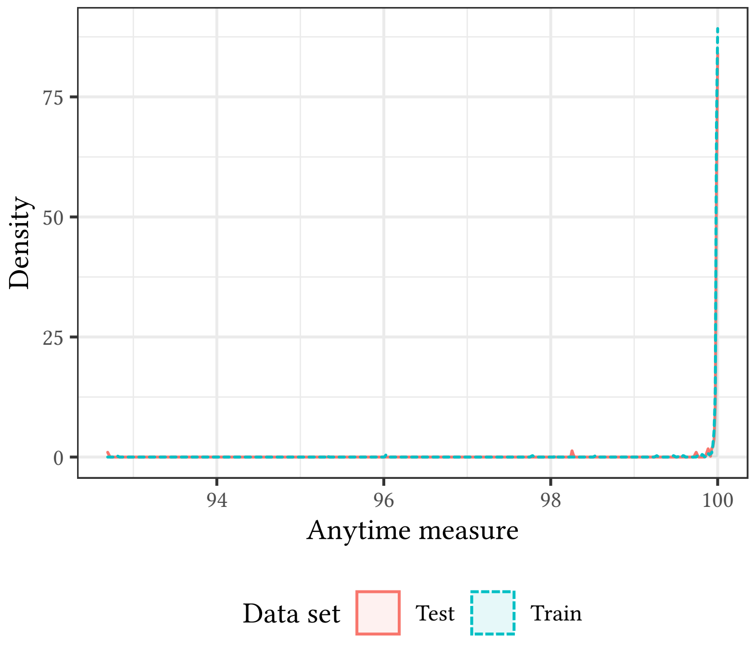


Figure 4.31: Anytime measure density plot for training and testing problem instances. GEPS, 2 objectives.

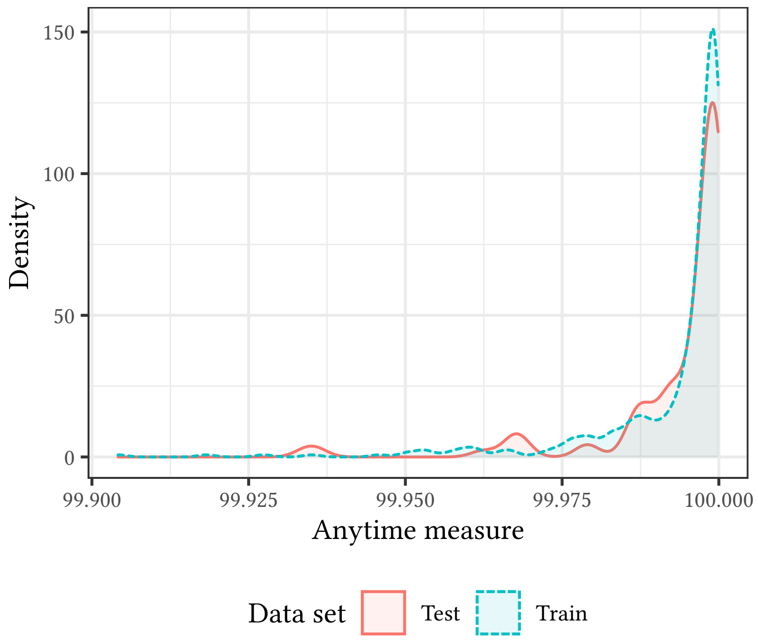


Figure 4.32: Anytime measure density plot for training and testing problem instances, for anytime measures greater than 99.9. GEPS, 2 objectives.

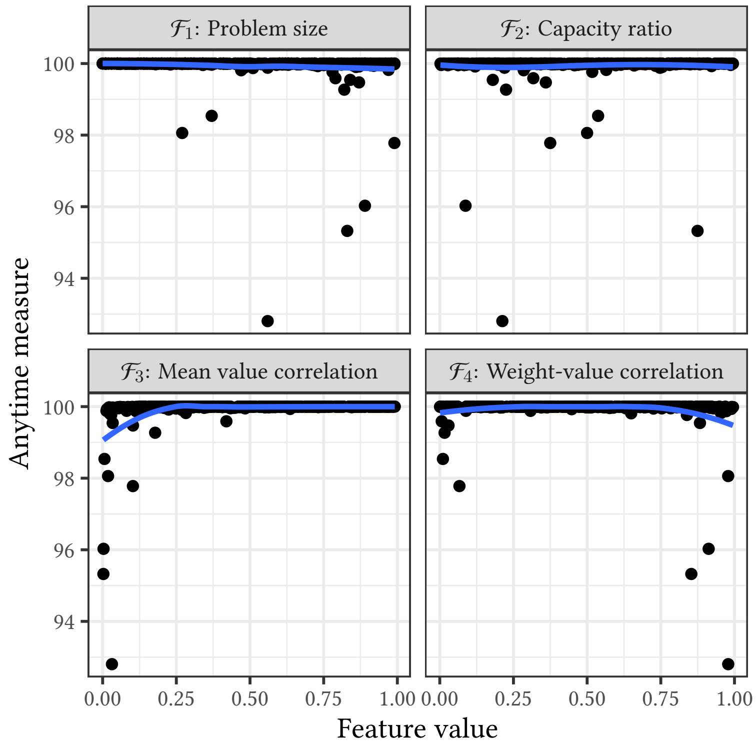


Figure 4.33: Relation between anytime measure and feature values for training problem instances. GEPS, 2 objectives.

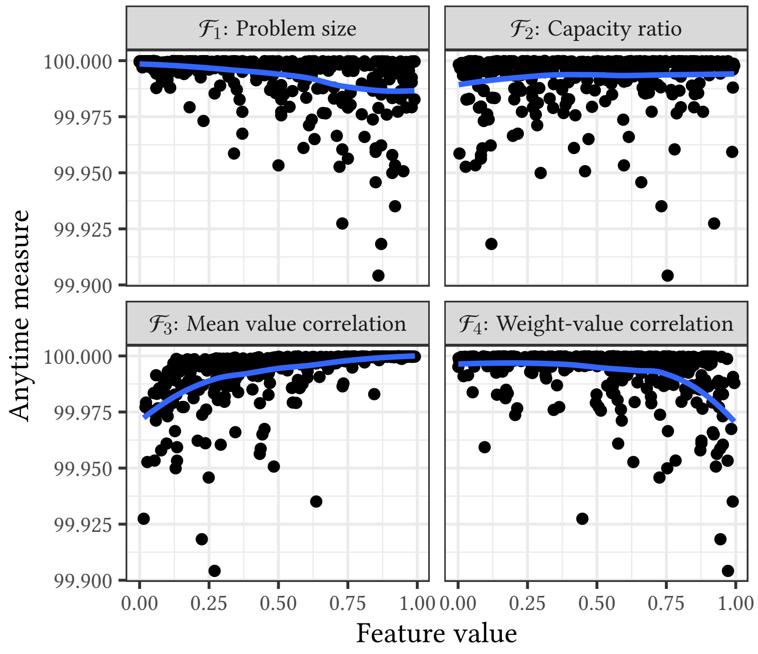


Figure 4.34: Relation between anytime measure and feature values for training problem instances, for anytime measures greater than 99.9. GEPS, 2 objectives.

$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
-0.385	0.131	0.798	-0.297

Table 4.25: Spearman's correlation coefficient between the features and anytime measures of the training instances. GEPS, 2 objectives.

ξ_1	ξ_2	ξ_3	ξ_4
4.667	1.689	24.361	3.286

Table 4.26: Weights for the distance metric. GEPS, 2 objectives.

3	4	5	6	7	8	9	10	11	12
0.080	0.082	0.077	0.075	**0.074**	0.075	0.076	0.078	0.078	0.078

Table 4.27: MAE given by LOOCV on the training problem instances. GEPS, 2 objectives.

Figure 4.35 gives the histogram of prediction errors on all testing problem instances. As expected, the errors are very small for almost every instance, with a few exceptions where anytime measure is significantly over-estimated.

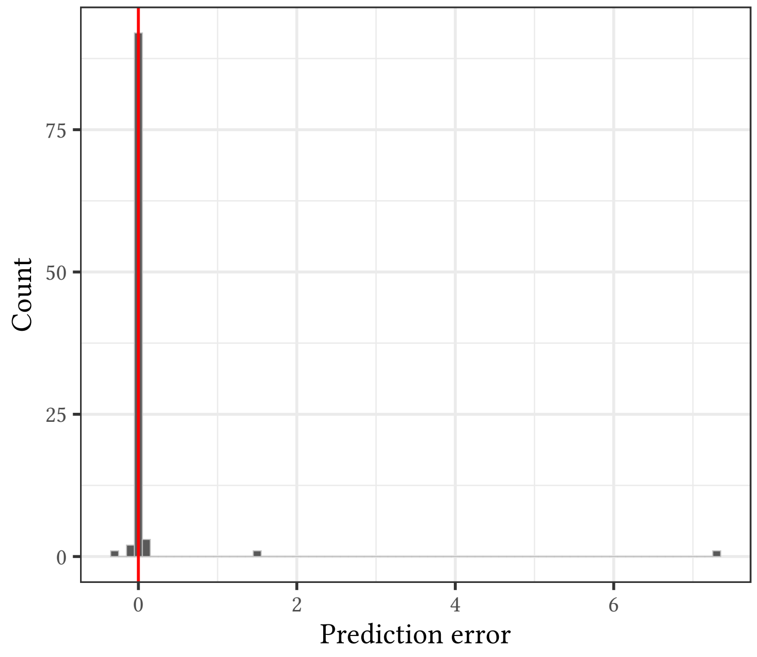


Figure 4.35: Histogram of the prediction error in terms of anytime measure for testing problem instances. The red line corresponds to the median. GEPS, 2 objectives.

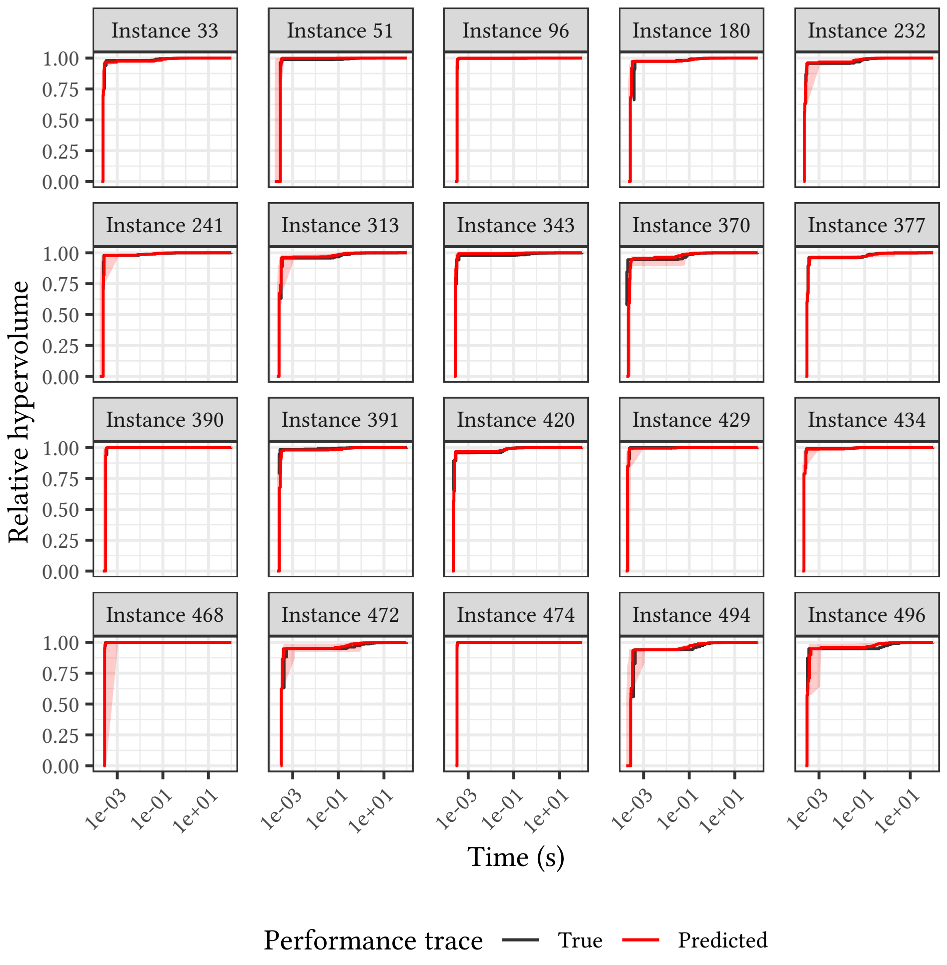


Figure 4.36: True and predicted anytime performance for 20 randomly selected testing problem instances. GEPS, 2 objectives.

Figure 4.36 gives the true and predicted anytime performance for 20 randomly selected testing problem instances. The results for all testing problem instances can be found in Appendix A.7. We see that, in every case the predicted anytime performance follows the true anytime trace, despite the fact that these instances have slightly different anytime behavior, especially for small CPU times. This shows that our model, does not only give small prediction errors because the differences between the instances are small, but that it can in fact predict anytime performance quite well.

Figure 4.37 and Table 4.28 show the true and predicted anytime measure and anytime performance for the testing problem instances with the largest prediction errors. We see that the first two instances have a significant error, but also that these are the outliers that we expected at the beginning of this section. Besides those two instances, the anytime performance of the other 8 instances is predicted quite accurately.

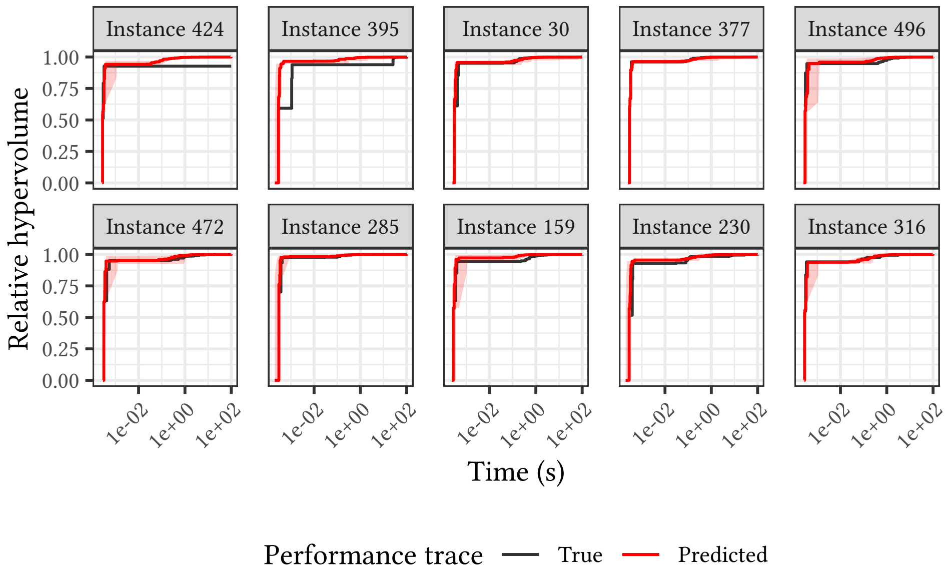


Figure 4.37: True and predicted anytime performance for testing problem instances that resulted in the largest prediction errors. GEPS, 2 objectives.

Instance	Predicted measure	True measure	Error
424	99.98	92.69	7.29
395	99.79	98.26	1.53
30	99.66	99.98	-0.31
377	99.88	99.99	-0.10
496	99.97	99.89	0.08
472	99.94	99.88	0.06
285	99.94	99.99	-0.06
159	99.99	99.93	0.05
230	99.79	99.74	0.05
316	99.97	99.94	0.04

Table 4.28: True and predicted anytime measure values for testing problem instances that resulted in the largest prediction errors. GEPS, 2 objectives.

4.3 Discussion

In this chapter, we started by presenting three distinct frameworks for creating empirical models to predict the anytime behavior of algorithms on previously unseen problem instances, taking into account empirical anytime performance data collected a priori. Although our goal is to predict anytime performance for the task of algorithm selection, these empirical models can easily be applied to other tasks such as automated algorithm configuration and anytime performance monitoring.

We also performed an experimental study for a model implemented according to the first framework, in which we analyzed its performance for predicting the anytime behavior of the PLS, BHV-DP, and GEPS algorithms on MOBKP instances with 2, 3, and 5 objectives considering 4 simple instance features. The results were very good for PLS and GEPS algorithms on problem instances for all numbers of objectives considered, and for the BHV-DP algorithm on problem instances with 2 objectives. In particular, we were able to accurately predict the anytime measure and anytime behavior of almost all testing problem instances. The results for the BHV-DP algorithm on problem instances with 3 and 5 objectives were not as good, and there were a considerable number of instances where the model failed to predict the anytime measure and anytime behavior with enough precision. However, we consider that the issue is not with the model itself, but rather the number of training instances available considering the range of parameters that we considered for generating instances. As such, it would be relevant to consider a more thorough study for the prediction of anytime behavior for the BHV-DP algorithm on whether more instances could improve the quality of the prediction.

Nonetheless, we consider that the resulting models obtained in the experimental study can be used for the task of algorithm selection, since they can adequately predict the tendency in anytime behavior in most cases, which might be sufficient for many selection problems. Therefore, we will consider these models in the following chapter.

For the future, it would be relevant to perform a similar experimental study for models implemented according to the other frameworks, or even for different models following the first framework, e.g., considering a different learning technique to find similar instances or even to consider manually categorized instances a priori. Preliminary experiments for a model following the second framework, showed that it can accurately model and predict anytime behavior. However, one difficulty that arose was in choosing an appropriate growth curve to model the anytime behavior of the algorithms. During these preliminary experiments we considered a Weibull growth curve, which worked reasonably well for many cases, especially for problem instances with 2 and 3 objectives. However, as the number of objectives grew we saw that perhaps a different growth curve was needed. Moreover, when taking into account within-group correlation and variance, we found that estimating the model parameters using the Generalized Non-Linear Least Squares (GNLS) required very good initial estimates, otherwise, the GNLS method often failed to find the model parameters. Still, this is something that would be interesting to explore further, as initial results were quite promising.

Finally, it would be interesting to consider the modeling of different problems and algorithms using these models. Other interesting topics include taking into account other instance features, and to analyze the impact of different instance features on prediction quality.

Chapter 5

Offline Algorithm Selection

In this chapter, we consider the development of an automated offline algorithm selection methodology for selecting between MOO algorithms such that the anytime performance of the selected algorithm is optimal with respect to the *anytime preferences* of the DM. This methodology takes into account models of anytime performance to predict the anytime performance profile for a previously unseen problem instance. In the previous chapters we presented both a theoretical model and an empirical model. In this chapter we will only consider the later since we want to consider anytime performance in terms of CPU-time.

We assume that the anytime preferences of the DM are known at the time of selection, and given to the algorithm selection methodology. Moreover, we consider that the anytime preferences of a DM are given by a general utility function $U(t, q) \to \mathbb{R}_{\geq 0}$ that denotes how likely the chosen algorithm is to be interrupted at a particular time t and archive quality q. The offline algorithm selection problem is then defined as:

$$\operatorname*{argmax}_{a \in \mathcal{A}} M(a, \iota, U) \tag{5.1}$$

where $\mathcal{A}$ denotes the set of available algorithms, ι denotes the previously unseen instance for which we want to select an algorithm, and $M(a, \iota, U) \to \mathbb{R}$ denotes a scalar measure of anytime performance for an algorithm a on instance ι with respect to a utility function U that, without loss of generality, is to be maximized. Note that, we cannot generally expect to have an exact anytime performance measure for a previously unseen instance. As such, we will consider approximating the anytime performance measure.

The remainder of this chapter is organized as follows. In Section 5.1, we give a performance measure to characterize the anytime performance of an algorithm as a scalar value. In Section 5.2, we give a general offline algorithm selection methodology based on the empirical models of the previous chapter.

In Section 5.3, we carry out an experimental study to analyze the quality of the proposed methodology for selecting between anytime algorithms to the MOBKP. The results indicate that our approach can very often select the best algorithm. Lastly, in Section 5.4 we summarize the findings of this chapter and discuss possible directions for future work.

5.1 Anytime Performance Measure

We start by defining a pre-order that denotes whether a performance profile is at least as good as another with respect to a utility function U that denotes the anytime preferences of the DM.

Definition 5.1 ($\geq_U$ pre-order)**.** Given two performance profiles P and P', and a utility function $U(t,q) \to \mathbb{R}_{\geq 0}$, we define the relation $P \geq_U P'$ iff $P(t,q) \geq P'(t,q)$ holds for every t and q where $U(t,q) > 0$.

Note that, for two algorithms a, b with performance profiles P_a and P_b respectively, a relation $P_a \geq_U P_b$ means that algorithm a is equally likely or more likely than algorithm b to return an archive with quality q at time t for every t, q such that $U(t,q) > 0$. However, there may be problem instances and/or runs for which algorithm b is able to achieve a better archive quality q than algorithm a at a given time t.

The following binary relations between two performance profiles P and P' result from the $\geq_U$ pre-order:

$$\begin{aligned} P =_U P' &\iff P \geq_U P' \land P' \geq_U P && \text{(Equivalence)} \\ P >_U P' &\iff P \geq_U P' \land P' \not\geq_U P && \text{(Superiority)} \\ P \parallel_U P' &\iff P \not\geq_U P' \land P' \not\geq_U P && \text{(Incomparability)} \end{aligned}$$

It is desirable that a performance measure is order-preserving with respect to the $\geq_U$ pre-order. Moreover, it is relevant that it allows to distinguish between incomparable performance profiles, that is, the performance measure can give distinct values for incomparable performance profiles.

In the following we assume, without loss of generality, that the domain for time is given by $\mathbb{R}_{\geq 0}$, and that archive quality is given by a finite scalar value in $\mathbb{R}$ to be maximized that is less than or equal to q_u. Moreover, we consider that the positive values for our utility function U are bounded by a finite upper bound on time t_u and a finite lower bound on archive quality q_ℓ, that is:

$$U(t,q) \to \begin{cases} \mathbb{R}_{\geq 0} & \text{if } t \leq t_u \land q \geq q_\ell \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

Then, a performance measure for anytime performance taking into account the utility function U is given by:

$$M(P, U) = \int_{q_\ell}^{q_u} \int_{0}^{t_u} U(t, q)\, P(t, q)\, dt\, dq \tag{5.3}$$

We assume that $M(P, U)$ is well-defined, i.e., $U(t, q)\, P(t, q)$ is integrable on the interval given by the specified bounds of time and quality.

Similar measures can be derived for discrete time and/or quality domains [36]. Note that, López-Ibáñez and Stützle [51] considered the use of the weighted hypervolume for measuring anytime performance, which gives an equivalent measure for algorithms with monotonic behavior. However, considering our definition of a performance profile given in Chapter 2, then our measure gives a different result for algorithms with non-monotonic behavior.

In the following, we give the properties of our measure with respect to the $\geq_U$ pre-order described earlier.

Proposition 5.1. Performance measure $M(\cdot, U)$ is order preserving with respect to the $\geq_U$ pre-order, that is:

$$P \geq_U P' \implies M(P, U) \geq M(P', U) \tag{5.4}$$

Proof. If $P \geq_U P'$, then the following implications hold:

$$U(t, q) > 0 \implies U(t, q)P(t, q) \geq U(t, q)P'(t, q) \tag{5.5}$$

$$U(t, q) = 0 \implies U(t, q)P(t, q) = U(t, q)P'(t, q) \tag{5.6}$$

Then if both $M(P, U)$ and $M(P', U)$ are well-defined, which we have assumed to be, then the above implies that $M(P, U) \geq M(P', U)$. □

It can also be shown that our measure is strictly order-preserving, that is:

$$P >_U P' \implies M(P, U) > M(P', U)$$
$$P =_U P' \implies M(P, U) = M(P', U)$$

Proposition 5.2. Performance measure $M(\cdot, U)$ allows to distinguish between incomparable performance profiles, that is:

$$P \parallel_U P' \not\Rightarrow M(P, U) = M(P', U) \tag{5.7}$$

Proof. Let $t_u = 1$, $q_\ell = 0$, $q_u = 1$, and:

$$U(t, q) \rightarrow \begin{cases} 1 & \text{if } 0 \leq t \leq 1 \wedge 0 \leq q \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

$$P(t,q) \rightarrow \begin{cases} 0.4 & \text{if } 0 \leq t \leq 0.5 \wedge 0 \leq q \leq 1 \\ 0.6 & \text{if } t > 0.5 \wedge 0 \leq q \leq 1 \end{cases} \tag{5.9}$$

$$P'(t,q) \rightarrow \begin{cases} 0.3 & \text{if } 0 \leq t \leq 0.5 \wedge 0 \leq q \leq 1 \\ 0.9 & \text{if } t > 0.5 \wedge 0 \leq q \leq 1 \end{cases} \tag{5.10}$$

Then $M(P,U) = 0.5$ and $M(P',U) = 0.6$, and it holds that $P \parallel_U P'$ and $M(P,U) \neq M(P',U)$. □

5.2 Anytime Algorithm Selection

As discussed in Section 2.5, existing selection methodologies can be divided into two categories, regression and classification approaches. However, existing methodologies fail to consider the anytime preferences of the DM at the time of selection. In this section, we consider how the anytime preferences can be included in an algorithm selection methodology.

Assuming static anytime preferences, i.e., that anytime preferences do not change between calls to the selection methodology, then a regression approach could learn to predict the anytime performance measure for the algorithms from a set of training instances, e.g., by running the algorithms against the training instances and computing the anytime performance measure for the given static anytime preferences. If the anytime preferences are instead given to the algorithm selection methodology then one alternative would be to train a similar model, but to compute the anytime performance measure against several training utility functions, which should represent the true utility functions that may appear during selection. If the set of expected utility functions is well known and if the type of the utility function is easily identifiable, then we can train multiple models, one for each type of utility function. If instead, the utility functions are not easily identifiable, we may consider using a set of features related to the utility function, e.g., the time and quality bounds. However, this is likely to quickly become infeasible as we move to consider more general utility functions, since we need to introduce more features and more training utility functions.

In a classification approach, similar issues can arise. However, these may be somewhat limited by the fact that the classification approach does not need to learn to predict the quality. For example, consider the comparison of two algorithms: a heuristic that quickly achieves a good solution but stops on a local optimum, versus an exact algorithm that slowly starts with worse solutions but eventually reaches the global optimum. Consider that, for each instance, there is a time point t such that the exact algorithm is always better than the heuristic after t. If the classification system learns to predict that after t the exact algorithms is always better, then for any utility function with a lower bound on time greater than t the exact approach is better, and there is no need to consider such utility functions, which may help reducing training times. However, a larger number of utility functions and/or features may still be required for the remaining cases.

The issues discussed above mostly stem from the fact that we are trying to incorporate the utility functions into the training phase. Therefore, we consider an alternative regression approach that first learns to predict the anytime performance profile for the instance that we are trying to select an algorithm for, and only takes into account the utility function when computing the anytime performance measure to perform the selection. More generally, we describe this selection approach as follows:

1. Given a set of algorithms $\mathcal{A}$ and a set of training instances learn to predict the performance profile of an algorithm $a \in \mathcal{A}$ for a previously unseen instance.

2. Given a previously unseen instance ι, predict the approximate performance profile $\widetilde{P}_{a,\iota}$ for each algorithm $a \in \mathcal{A}$.

3. Given a utility function U, select the algorithm that, without loss of generality, maximizes $M(\widetilde{P}_{a,\iota}, U)$, i.e.:

$$\operatorname*{argmax}_{a\in\mathcal{A}} M(\widetilde{P}_{a,\iota}, U) \tag{5.11}$$

   Note that, $M(\widetilde{P}_{a,\iota}, U)$ is an approximation to $M(a, \iota, U)$.

Throughout this thesis we have already discussed everything that is required for this methodology. In particular, in Chapter 3 and Chapter 4, we proposed theoretical and empirical models to predict the anytime performance profile of an anytime algorithm for a previously unseen instance, and in Section 5.1 we proposed an anytime performance measure that takes a performance profile and utility function. In the next section, we implement an algorithm selection methodology using the empirical model developed in Chapter 4 for predicting anytime performance of anytime algorithms for MOBKP instances, and the anytime measure introduced in Section 5.1.

5.3 Experimental Study

In this section, we carry out an experimental study to analyze the quality of the algorithm selection approach proposed in the previous section. In particular, we compare our proposed approach against a random approach that randomly selects the algorithm to use, and against approaches that always select the same algorithm. This section is organized as follows. In Section 5.3.1 we describe the experimental setup. In Section 5.3.2 we show and analyze the results.

5.3.1 Experimental Setup

We consider the same instances, algorithm configurations, and runs that were used for Section 4.2.4. In particular, we consider 500 instances for each number of objectives $m \in \{2, 3, 5\}$, with the following parameter sampling:

- $n \in \mathcal{U}(50, 150)$;
- $\rho_v \in \mathcal{U}\left(\frac{6}{\pi}\sin^{-1}\frac{1}{2(1-m)}, 1\right)$;
- $\rho_w \in \mathcal{U}(-1, 1)$;
- $\omega \in \mathcal{U}(0.3, 0.7)$.

For each instance, we record the anytime performance trace, in terms of CPU-time and hypervolume, for one run of the BHV-DP and PLS algorithms for $m \in \{2, 3, 5\}$ objectives, and the GEPS algorithm for $m = 2$ objectives. We consider a timeout of 100 seconds for algorithm execution. The algorithms are configured as follows:

- BHV-DP: ordering of the items following $O^{\min}$;
- PLS: default ordering of the items, initial empty solution $x = \{0, \ldots, 0\}$, random selection of the next item to process;
- GEPS: Parameter $\ell = 100$ to build the theoretical model that guides this approach.

The `C++` implementation for the algorithms is available at [33]. The code used to generate the problem instances, and to gather the performance traces of the algorithms, is available at [32].

We consider two selection scenarios with distinct utility functions. First, we consider a scenario where there is almost complete uncertainty in terms of when the algorithm will be interrupted, with the exception that it will be stopped before or at time t_u. Moreover, we consider that there is no anytime preference with regards to solution quality, and that solution quality is defined in the range $[0, 1]$. In this first scenario, the utility function is given by:

$$U_1(t, q) \rightarrow \begin{cases} 1 & \text{if } 0 \leq t \leq t_u \wedge 0 \leq q \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

The second scenario considers that a time t^* is known such that the algorithm will be interrupted at exactly that time. Moreover, we consider that there is no anytime preference with regards to solution quality, and that solution quality is defined in the range $[0, 1]$. In this second scenario, the utility function is given by:

$$U_2(t, q) \rightarrow \begin{cases} 1 & \text{if } t = t^* \wedge 0 \leq q \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

For each scenario, we consider 200 values of t_u and t^* evenly distributed in the range $[1\text{e}{-4}, 1\text{e}2]$ on a log scale.

For our proposed algorithm selection approach, we consider the empirical models that were trained in Section 4.2, i.e., which correspond to our implementation of the first empirical model (Section 4.1.1). Moreover, we consider the same split between testing and training instances that was used in Section 4.2, such that the same training instances are used to train the model, and the testing instances are used for testing our proposed approach. Lastly, if more than one algorithm maximizes the anytime performance measure, the approach selects one of them at random.

5.3.2 Results

The following sections show the results of algorithm selection using our proposed approach. In each section we present the results for both utility functions considering instances with a particular number of objectives. The code to reproduce the experiments is available at [32].

2 Objectives — 3 Algorithms

In this section, we analyze the automated selection between the PLS, BHV-DP, and GEPS algorithms for problem instances with 2 objectives.

Table 5.1 gives the accuracy, with respect to the ratio of correct algorithm selections, on the first selection scenario for five distinct selection approaches: our proposed approach, a random approach that randomly selects one of the algorithms, and three other approaches, each of which corresponds to the selection of a single algorithm at all times as indicated by its name in the table. This ratio takes into account selection problems for all 100 testing problem instances and 200 values of t_u. We see that our approach has an accuracy value equal to 0.97, which is the highest, and indicates that our approach can very often choose the best algorithm. The random approach has a accuracy value close to $1/3$, which is to be expected since there are three algorithms for

Proposed	Random	PLS	BHV-DP	GEPS
0.97	0.33	0.07	0.01	0.92

Table 5.1: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the first scenario. 2 objectives, 3 algorithms.

selection. Always selecting the GEPS gives an accuracy equal to 0.92, whereas the other two algorithms have very low accuracy, which means that the GEPS is very often the best choice.

Figure 5.1 summarizes the accuracy for the first selection scenario over all testing instances for each setting of t_u. We see that after the first values of t_u, our approach always selects the GEPS algorithm, since it follows the line of the approach that always select the GEPS algorithm. Moreover, we see that for very small values of t_u, our proposed approach often selects the PLS algorithm, since it closely follows the line of the approach that always selects the PLS algorithm. Selecting the PLS algorithm is clearly a good choice for the first few values of t_u. However, as the values of t_u increase slightly, i.e., for values

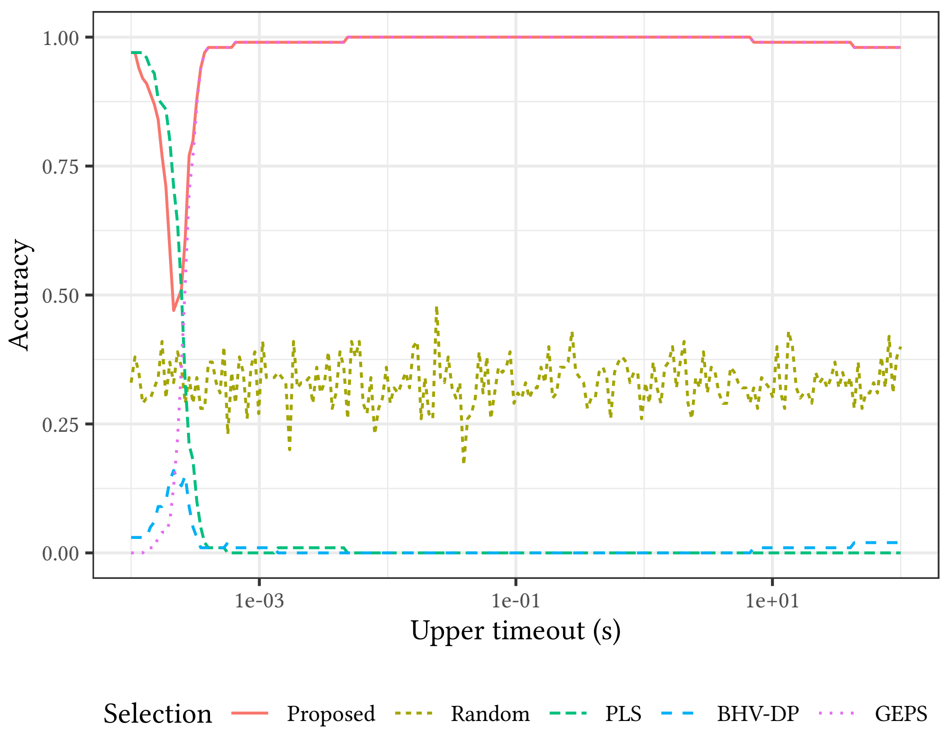


Figure 5.1: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the first scenario at each value of t_u. 2 objectives, 3 algorithms.

Proposed	Random	PLS	BHV-DP	GEPS
0.88	0.40	0.06	0.41	0.72

Table 5.2: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the second scenario. 2 objectives, 3 algorithms.

$t_u \in [2.1\text{e}{-4}, 2.5\text{e}{-4}]$, our approach has an accuracy close to 0.5. Further analysis of the data revealed that our approach tends to wrongly select the GEPS algorithm for those values of t_u, when it should be selecting the BHV-DP or PLS algorithms instead. The full prediction results in Appendix A show that the prediction model for the GEPS algorithm often overestimates the solution quality obtained by the algorithm at such small CPU-time values. As such, to improve the accuracy of our proposed selection approach in this scenario we should focus on improving the prediction at small CPU-time values for the GEPS algorithm.

Table 5.2 gives the accuracy for the second selection scenario, i.e., with utility function U_2, and the same five distinct selection approaches. It summarizes the accuracy over the 100 testing instances and 200 values of t^*. We see that our approach got the highest accuracy with a value equal to 0.88, which is slightly worse than before, but still quite good. Note that, the random approach achieved a value slightly larger than $1/3$. This happens because for values of t^* close to 100, both the BHV-DP and the GEPS algorithms have found a solution with optimal solution quality value, and as such, choosing any of those algorithms is correct. We also see that always selecting the GEPS algorithm now has slightly worse accuracy, and always selecting the BHV-DP algorithm has improved significantly. This is due to the fact that the BHV-DP algorithm can always find the Pareto set within the 100 second timeout, as such, it will often be the best choice for larger values of t^*.

Figure 5.2 summarizes the accuracy for the second selection scenario over all testing instances for each setting of t^*. For very small values of t^* the behavior is similar to that of the first scenario, that is, our approach can often select the PLS correctly for very small values of t^*, but then wrongly selects the GEPS algorithm for values of t^* close to $2\text{e}{-4}$, instead of the BHV-DP and PLS algorithms. However, what is different is the behavior for values of t^* greater than $1\text{e}{-2}$. For these values, the GEPS is not always the best choice like in the first scenario, and we see that the BHV-DP steadily becomes the best choice. In this case, we see that when two choice lies between two algorithms, our approach can often make the correct selection. However, the fact that

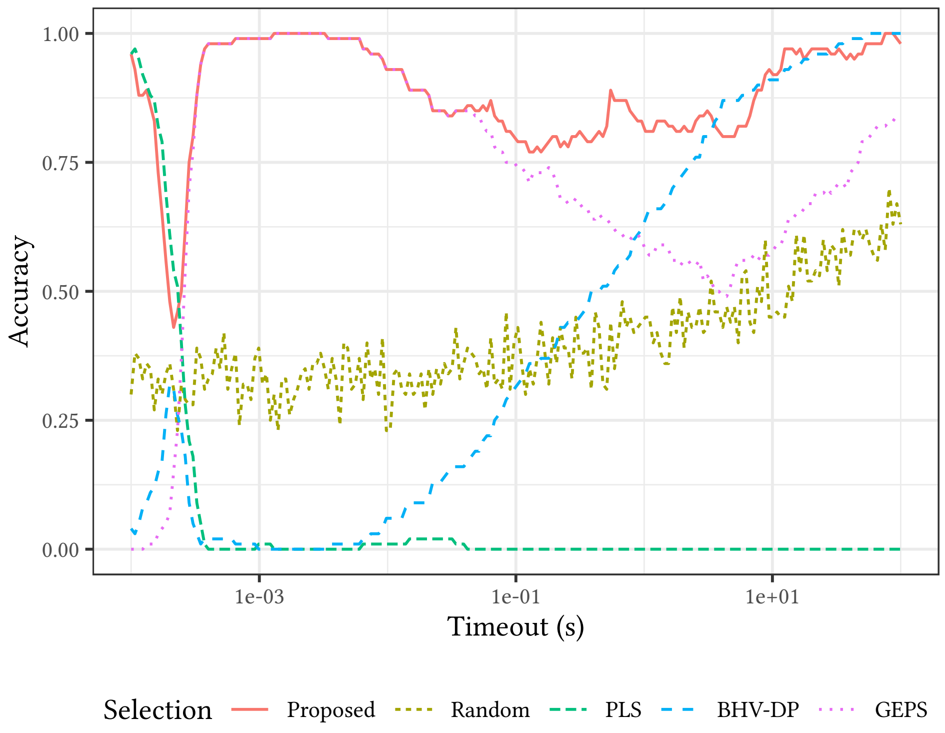


Figure 5.2: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the second scenario at each value of t^*. 2 objectives, 3 algorithms.

accuracy is close to 0.75 at times, indicates that there is still some room for improvement. The prediction results for the BHV-DP and GEPS algorithms, which can be seen in Appendix A.4 and Appendix A.7 respectively, indicate that the issue is the prediction of anytime performance for the BHV-DP algorithm, which is sometimes not very accurate for CPU-time values greater than 1e−2. As such, to improve the selection in this case, we would primarily need to improve the prediction model of the BHV-DP algorithm.

2 Objectives — 2 Algorithms

In the previous section, we saw that the GEPS algorithm was very often better than the other two algorithms, in particular in the first scenario. Moreover, the PLS was only the best choice for very small values of t_u and t^*. In this section, we choose to ignore the GEPS algorithm, to better analyze the selection between the PLS and the BHV-DP algorithms for instances with 2 objectives.

Table 5.3 gives the overall accuracy on the first selection scenario. Our proposed approach got an accuracy value equal to 0.90, which was the highest. This is slightly less than in the previous case. However, given that always selecting the PLS or BHV-DP algorithms resulted in accuracy values equal

Proposed	Random	PLS	BHV-DP
0.90	0.51	0.63	0.37

Table 5.3: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the first scenario. 2 objectives, 2 algorithms.

to 0.63 and 0.37 respectively, the accuracy of our proposed approach looks particularly good, and indicates that our approach is consistently selecting the best algorithm.

Figure 5.3 gives the accuracy on the first selection scenario for different values of t_u. We see that the PLS algorithm is often the best choice for smaller values of t_u, whereas the BHV-DP algorithm is often the best choice for larger values of t_u. As for our approach, we see that it can often select the best approach and that accuracy only falls slightly below 0.75 for small timeout values. The prediction results in Appendix A.1 and Appendix A.4, for the PLS and BHV-DP algorithms respectively, show that the prediction of the PLS algorithm is generally more accurate than that of the BHV-DP. As such, to improve

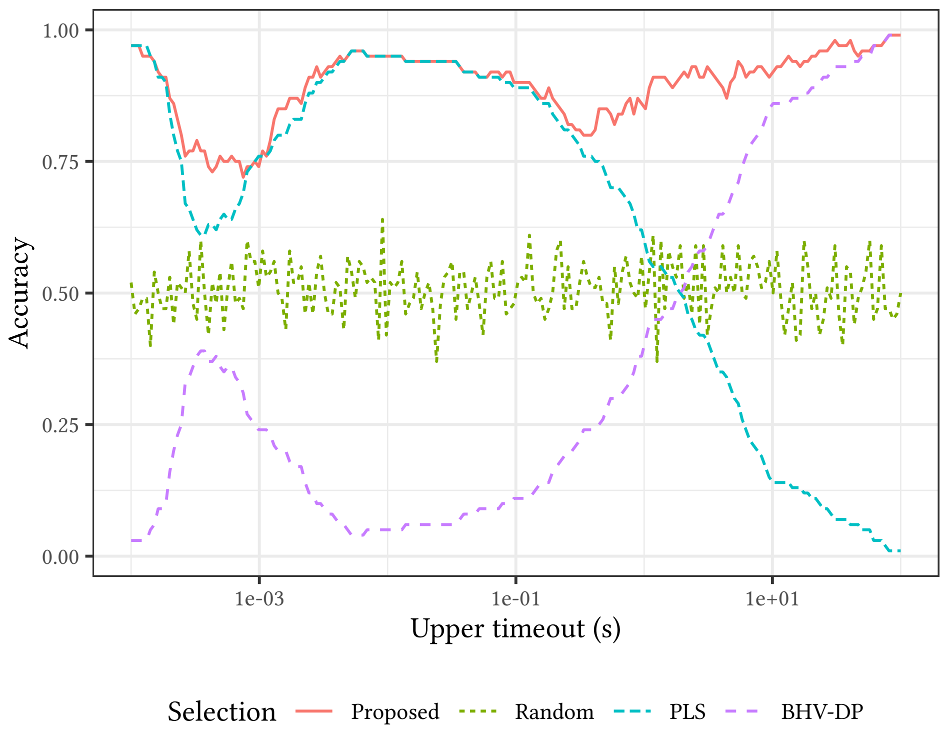


Figure 5.3: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the first scenario at each value of t_u. 2 objectives, 2 algorithms.

Proposed	Random	PLS	BHV-DP
0.90	0.50	0.52	0.48

Table 5.4: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the second scenario. 2 objectives, 2 algorithms.

the accuracy, we consider that the primary focus should be on improving the prediction of anytime performance for the BHV-DP algorithm.

Table 5.4 gives the accuracy on the second selection scenario. We see that the results are not very different than those in the first scenario, and that our approach still has the highest accuracy. However, the accuracy of choosing only the PLS or BHV-DP algorithms is now more balanced. Therefore, the fact that our proposed approach continues to get such an high accuracy, clearly indicates that it can often select the best algorithm.

Figure 5.4 gives the accuracy on the second selection scenario for different values of t^*. We see a very similar behavior when compared to the first scenario. In particular, we see that the PLS algorithm is often the best choice for small

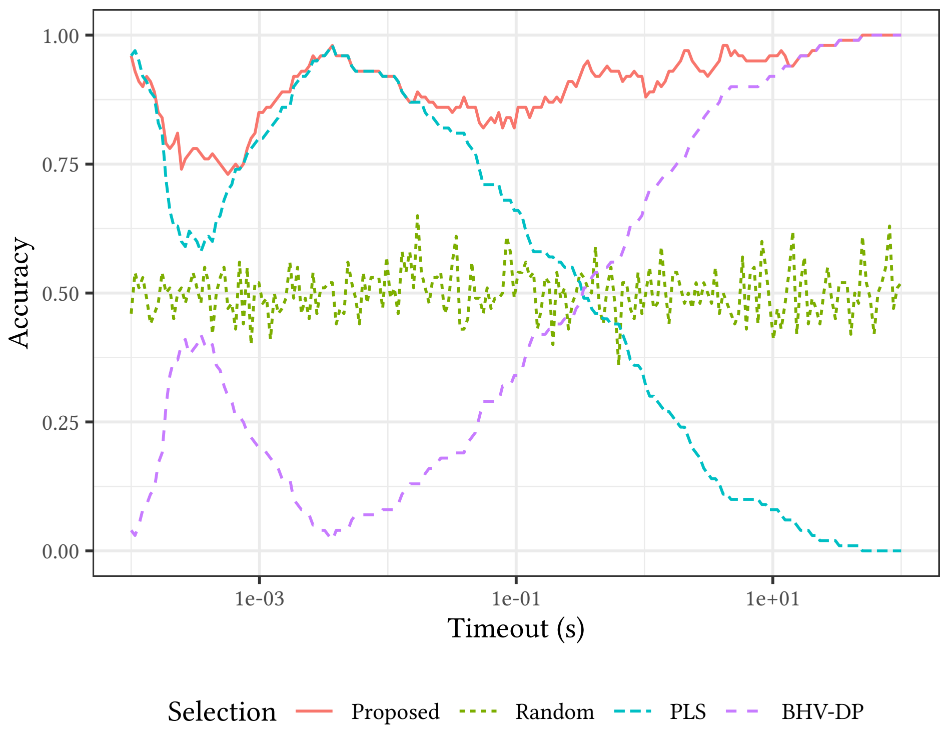


Figure 5.4: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the second scenario at each value of t^*. 2 objectives, 2 algorithms.

values of t^*, whereas the BHV-DP algorithm is often the best choice for large values of t^*. Moreover, we see that our approach can often select the best algorithm, and that accuracy almost never drops below 0.75. Nonetheless, there is still room for improvement, and, as before, we consider that to improve the accuracy of our approach the prediction of anytime performance for the BHV-DP algorithm should be improved.

3 Objectives — 2 Algorithms

We now turn to the selection between the PLS and BHV-DP algorithms on problem instances with 3 objectives.

Table 5.5 gives the accuracy on the first selection scenario considering all testing problem instances and values of t_u. We see that our proposed approach got an accuracy value of 0.87, which is the highest. Moreover, we see that choosing only the PLS algorithm had an accuracy value of 0.74, and choosing only the BHV-DP algorithm resulted in an accuracy value of 0.26.

Figure 5.5 gives the accuracy on the first scenario for each value of t_u. We see that our proposed approach is always equal to or better than the other approaches. However, we see that for values of t_u close to 1e−3 it has a significantly low accuracy compared to other values of t_u. The full prediction results for the PLS and BHV-DP algorithms, which can be seen in Appendix A.2 and Appendix A.5 respectively, show that the prediction for CPU-times around 1e−3 is generally quite good for both algorithms, save for a few exceptions. Moreover, for both algorithms the solution quality, in terms of relative hypervolume, at that time is often very close to 0. On the one hand, this shows that small errors in prediction can be significant for selection accuracy, and should not be discarded. On the other hand, the fact that quality values are so small indicate that perhaps this is not a very relevant selection scenario. For larger values of t_u, we see that the accuracy of our approach is quite good, i.e., with accuracy values around 0.875, despite the fact that there is no algorithm that is clearly the best. For large CPU-time values, the prediction results for the BHV-DP algorithm are clearly worse than for the PLS algorithm. Moreover, in this case, the differences between the true and the predicted values are more

Proposed	Random	PLS	BHV-DP
0.87	0.50	0.74	0.26

Table 5.5: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the first scenario. 3 objectives, 2 algorithms.

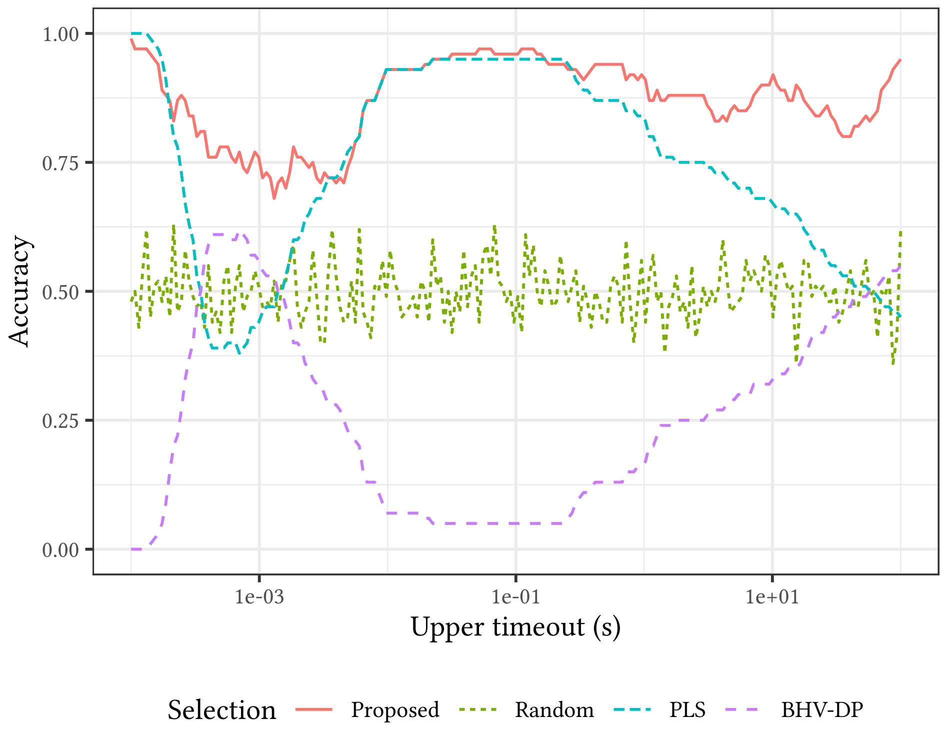


Figure 5.5: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the first scenario at each value of t_u. 3 objectives, 2 algorithms.

significant. Thus, the error in anytime measure due to a wrong selection is expected to be more significant. Therefore, we consider that to improve selection we should start by improving the predictions for the BHV-DP algorithm.

Table 5.6 and Figure 5.6 give the accuracy results for the second selection scenario. We see that the results are quite similar to the first selection scenario. As before, we consider that accuracy could be further improved, in particular, for larger values of t^*, if the prediction of anytime performance for the BHV-DP algorithm is improved.

Proposed	Random	PLS	BHV-DP
0.87	0.50	0.69	0.31

Table 5.6: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the second scenario. 3 objectives, 2 algorithms.

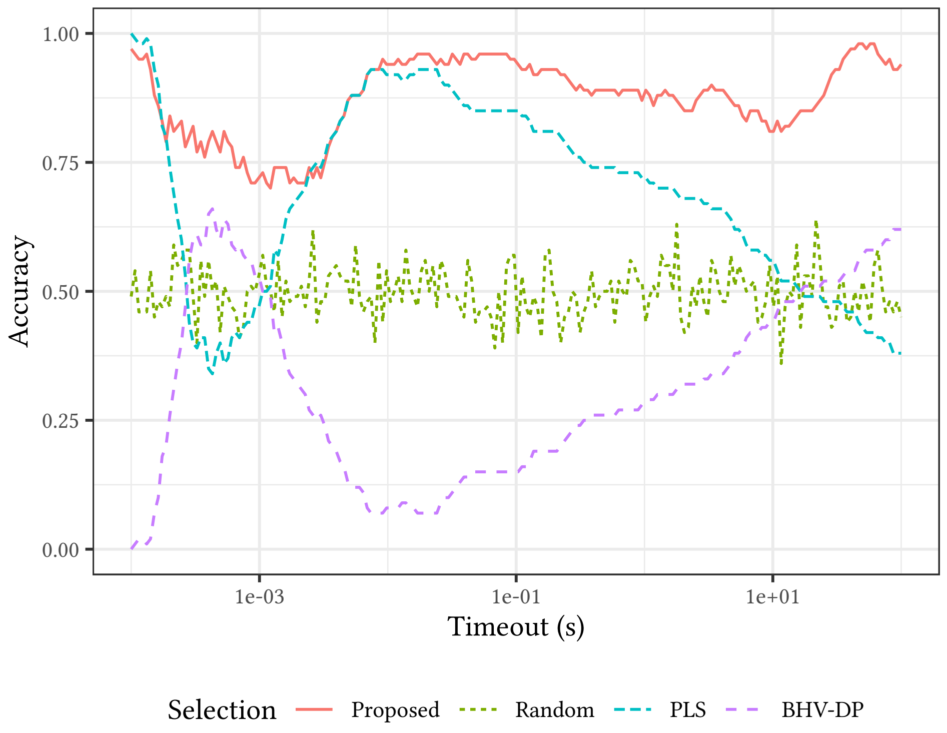


Figure 5.6: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the second scenario at each value of t^*. 3 objectives, 2 algorithms.

5 Objectives — 2 Algorithms

Lastly, we analyze the selection between the PLS and BHV-DP algorithms for problem instances with 5 objectives.

Table 5.7 gives the accuracy on the first selection scenario. We see that our proposed approach got an accuracy value of 0.87, whereas choosing only the PLS algorithm had an accuracy value of 0.69, and choosing only the BHV-DP algorithm resulted in an accuracy value of 0.31. These results indicate that our approach can often select the best algorithm.

Figure 5.7 gives the accuracy on the first selection scenario for each value of t_u. We see that our proposed approach is often better than the other approaches.

Proposed	Random	PLS	BHV-DP
0.87	0.50	0.69	0.31

Table 5.7: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the first scenario. 5 objectives, 2 algorithms.

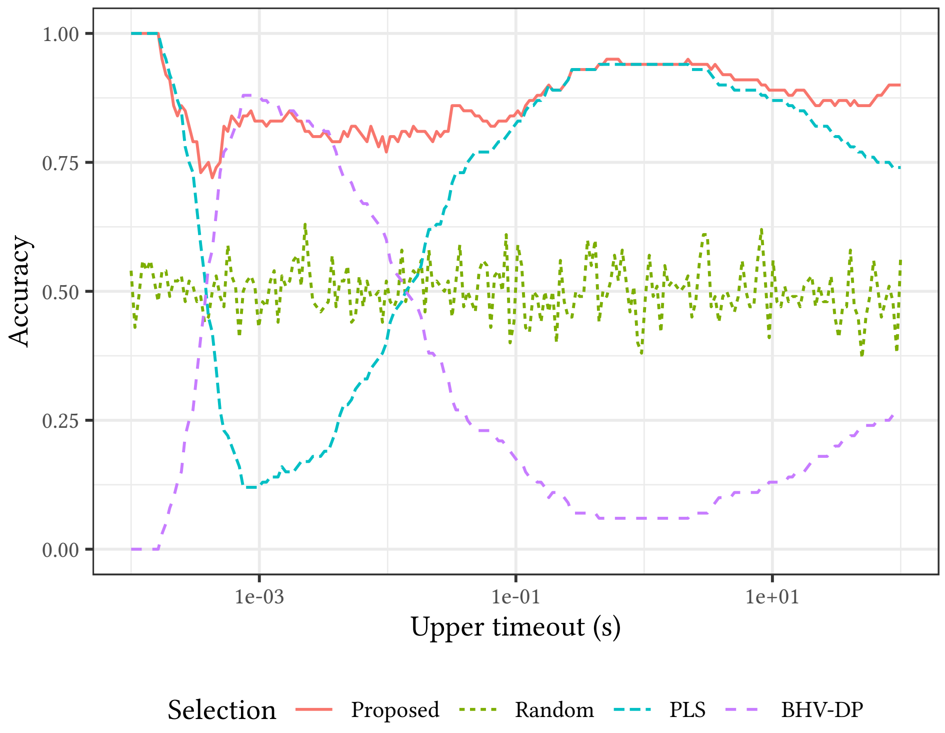


Figure 5.7: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the first scenario at each value of t_u. 5 objectives, 2 algorithms.

However, for values of t_u around 1e−3, we see that it is worse than always selecting the BHV-DP algorithm. Moreover, we get the worst accuracy at those values of t_u. Despite this, looking at the full prediction results for the PLS and BHV-DP algorithms, which can be seen in Appendix A.3 and Appendix A.6 respectively, we see that the prediction is quite good for both algorithms at small values of CPU-time, and that in both cases the solution quality, measured in terms of relative hypervolume, is close to zero. The latter observation indicates that perhaps this is not the most interesting selection scenario. On the other hand, for larger values of CPU-time, the prediction for the BHV-DP is worse than the PLS and the differences between the true and the predicted values are much larger. Therefore, to improve the selection we consider that it would be more relevant to focus on improving the prediction of anytime performance for the BHV-DP algorithm, in particular at larger CPU-times, so that not only is accuracy improved but also that the error when making a wrong selection is reduced.

Table 5.8 and Figure 5.8 give the accuracy results on the second selection scenario. We see that the results are quite similar to those of the first scenario. Therefore, we draw the same conclusions as before.

Proposed	Random	PLS	BHV-DP
0.86	0.50	0.67	0.33

Table 5.8: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the second scenario. 5 objectives, 2 algorithms.

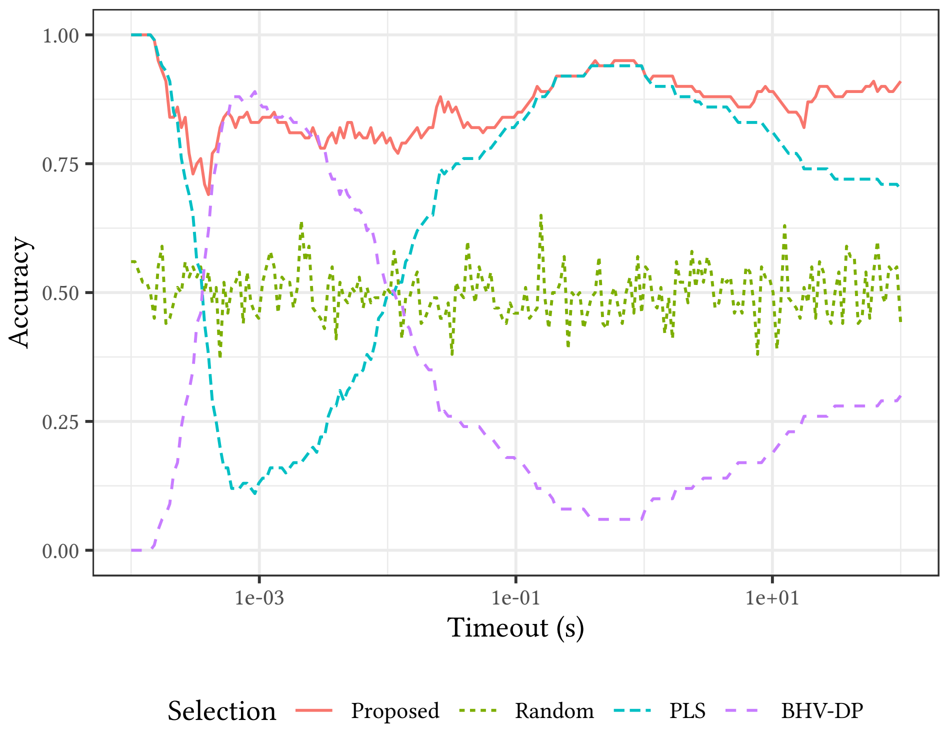


Figure 5.8: Accuracy of our proposed approach, a random approach, and approaches that always select the same algorithm, on the second scenario at each value of t^*. 5 objectives, 2 algorithms.

We remark that, despite the fact that the prediction of anytime performance for the BHV-DP algorithm on instances with 5 objectives often gives a large error, the selection accuracy remains quite high. The fact that the algorithms have such a different behavior is an important factor to this. Since otherwise, the models would need to be much more accurate. However, this is coupled with the fact that the predicted anytime performance is correctly capturing the tendency of anytime behavior, even when there is a significant error. Otherwise, we would not expect such a high accuracy.

5.4 Discussion

In this section we discussed a measure of anytime performance that can take into account the anytime preferences of a decision maker, and proposed an automated algorithm selection approach that can take these anytime preferences into account when selecting the best algorithm for a previously unseen instance. This algorithm selection approach considers models of anytime performance for the algorithms we want to select on previously unseen instances, such as those discussed in Chapters 3 and 4, as well as the discussed anytime measure to take into account the anytime preferences of a decision maker when selecting the best algorithm. Lastly, we carried out an experimental study to analyze the quality of this algorithm selection approach in selecting between algorithms to the MOBKP, considering two distinct utility functions.

This experimental study showed that our selection approach has very good accuracy, and generally works very well despite the fact that the models of anytime performance considered were, for some algorithms, not very accurate. Still, we highlighted several cases in which the accuracy of our approach left something to be desired due to the errors in the models of anytime performance. This shows that our approach relies on the quality of the selected models, and that to improve its accuracy we need to focus on improving the quality of the models. One relevant direction for future work is then to extend the experimental study to better analyze the relation between the models of anytime performance and the accuracy of our approach both theoretically and practically. For example, can lower and upper bounds for the accuracy of the selection approach be defined with respect to the quality of the empirical models, or how do changes to the empirical models such as using more instances and instance features affect our selection approach in practice.

Another aspect worth investigating is the time it takes for our approach to make a selection with respect to the empirical models being used. The way the code was developed did not allow to perform such measurements easily, and it was not developed with selection performance in mind. Still, preliminary experiments with the current code showed that computing the anytime performance profile and anytime measure for a single algorithm took about $1e{-2}$ seconds or less for the considered empirical model, which should be acceptable for most scenarios. Nonetheless, further research is needed to better study and report the computational effort needed to select an algorithm with respect to the empirical models being used.

To better put into context the performance of our approach, it would also be worth investigating other algorithm selection methodologies, to compare them to our approach. As previously discussed, one issue with more traditional classification and regression approaches is that they do not take into account the anytime preferences of the DM at the time of selection. Still, we could consider that the anytime preferences of a DM are fixed prior to training, thus allowing to compare with more traditional classification and regression approaches. Alternatively, it would be interesting to investigate the development of new methodologies that can take into account the anytime preferences of the DM at the time of selection, which would be comparable to our proposed approach. Lastly, it would be interesting to analyze the quality of the proposed algorithm selection approach in other problems, in particular, problems appearing in real-world scenarios.

Chapter 6

Online Algorithm Selection

In the previous chapter, we considered the selection of algorithms from an offline perspective, i.e., selecting the algorithm before starting its execution. In this chapter, we consider an online perspective for the ASP, that is, we want to select and change algorithms while already executing an algorithm to solve a previously unseen problem instance in order to improve anytime performance. In particular, we consider selecting between branch-and-bound strategies taking into account the anytime behavior of each strategy, and taking into account the performance of the strategy that is being executed.

To this end, we will start by proposing an Indicator-based Branch-and-Bound (IBB) approach that takes into account the potential of a node when selecting the next node to be explored in terms of a binary quality indicator [37], and give four selection strategies for this approach based on two distinct binary quality indicators. Then, we perform an empirical study to analyze the anytime performance of these selection strategies for the MOBKP, which shows that these approaches often have better anytime performance compared to traditional BB selection strategies. This empirical study also shows that not all selection strategies have the same anytime performance and that the selection strategy that most often gives the best approximation depends on the CPU-time and on the number of objectives of the problem instance being solved. Taking into account these empirical results, we propose an online methodology that changes between IBB selection strategies during execution based on a set of simple rules, with the goal of improving anytime performance. Lastly, we carry out an experimental study to analyze the anytime performance of this online methodology, which shows that it can often return better or equivalent approximations compared to the individual IBB selection strategies.

This chapter is organized as follows. In Section 6.1, we present the IBB approach and the implementation details of four selection strategies for this approach. We also present the results of the experimental study analyzing the anytime performance of these selection strategies. In Section 6.2, we propose and empirically analyze the online selection methodology to change between IBB selection strategies. In Section 6.3, we summarize the results of this chapter and discuss possible directions for future research.

6.1 Indicator-based Branch-and-Bound

6.1.1 Framework

As discussed in Section 2.3.2, BB approaches enumerate feasible solutions by recursively dividing the decision space such that each subdivision gives way to a subproblem. Each subdivision is often seen as a node in a search tree. A key aspect of BB approaches is that lower and upper bound sets can be computed for each node and used to avoid exploring nodes that cannot lead to efficient solutions.

An important component for a BB approach, which can greatly impact its (anytime) performance, is how to select the next node to be explored. In Section 2.3.2, we discussed four distinct selection strategies: DFS, which selects the nodes closest to the bottom of the search tree first; BFS, which selects the nodes closest to the top of the search tree first; BeFS, which selects the most promising node according to some heuristic criteria; BeDFS, which selects the most promising node that is closest to the bottom of the search tree. In this section, we are particularly interested in defining a heuristic to identify the most promising node in the BeFS and BeDFS strategies, in the context of MOO. In particular, we propose using binary quality indicators to measure the quality of the upper bound set of a node with respect to the archive of solutions kept by the BB algorithm. Two such indicators are the binary hypervolume indicator and the ε-indicator discussed in Section 2.2. We denote by Indicator-based Branch-and-Bound (IBB) a BB approach that uses BeFS or BeDFS strategies guided by quality indicators, as described in the following.

Formally, considering the BB frameworks described in Algorithms 1 and 2 in Section 2.3.2, and that, w.l.o.g., the binary quality indicator considered is to be maximized, a BeFS selection strategy can be implemented by defining the `SelectNode(Q)` function as follows:

$$\texttt{SelectNode}(Q) = \underset{\text{node}\in Q}{\operatorname{argmax}}\; I\left(U(\text{node}), \{f(x) : x \in S\}\right) \tag{6.1}$$

where Q denotes the active queue of nodes that can be selected, $U(\text{node})$ denotes the upper bound set of a node, S denotes the archive of solutions currently kept by the algorithm, and $I(A, B) \longrightarrow \mathbb{R}$ denotes a binary quality indicator that characterizes the quality of a set $A \subset \mathbb{R}^m$ with respect to a set $B \subset \mathbb{R}^m$. For a BeDFS selection strategy, the `SelectNode` is similar, but it considers only the nodes in Q that are at the deepest level of the search tree.

Note that, under the assumption that the binary quality indicator I is order-preserving for a fixed reference set, then this selection guarantees that the upper bound set of the chosen node is not dominated by the upper bound sets of the remaining nodes. Moreover, if the indicator is strictly order-preserving, and the optimal solution to the selection is unique, then the upper bound set of the chosen node is further guaranteed to not be weakly dominated by the upper bound sets of the remaining nodes. In Section 2.2 we have shown that the binary hypervolume is strictly order-preserving, and that the ε-indicator is order-reversing, which is functionally equivalent since, by definition, its negation is order-preserving.

As an example, in Figure 6.1 we illustrate the upper bound set $\{u\}$ of two distinct nodes, and in gray the area given by the binary hypervolume indicator for each upper bound set with respect to the image of the archive in the objective space (points in black). In this case, our selection strategy would select the node that gives the upper bound of the left-hand side figure.

A potential issue for this approach is that the computational cost of the binary quality indicator may be high, thus slowing down the search process significantly. For example, this can be an issue when considering the binary hypervolume indicator for problem instances with a large number of objectives [27]. We expect this to be less critical for the BeDFS strategy since it only considers the nodes that are closest to the bottom of search tree which are often small in number.

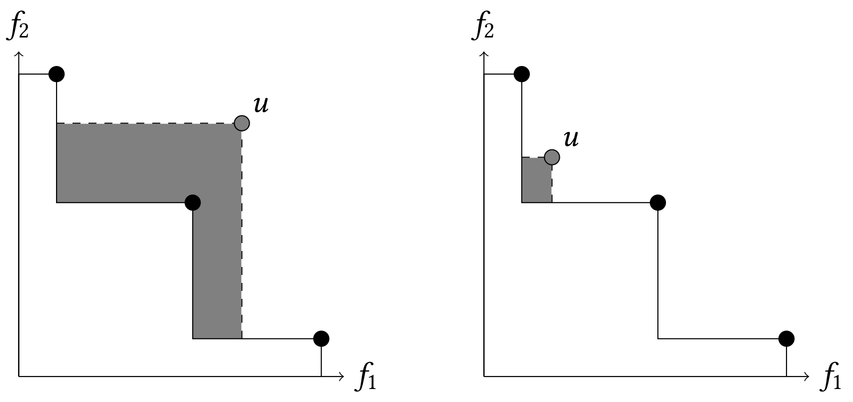


Figure 6.1: Illustration of the binary hypervolume indicator (in gray) for the upper bound set $\{u\}$ of two distinct nodes.

6.1.2 Implementation Details

In this section we discuss the components of an IBB approach to the MOBKP problems using the BeFS and BeDFS selection strategies. We consider that the bound sets are computed according to the strategy described in Chapter 2.

The binary hypervolume and the ε-indicator, described in Section 2.2, are considered as the binary quality indicators for the BeFS and BeDFS selection strategies. Since we consider an upper bound set comprised of a single point in the objective space, the binary hypervolume computation can be formulated as a hypervolume contribution calculation. For two objectives, the hypervolume contribution can be computed in $O(\log \mu)$ time, where μ is the number of solutions in the archive, if the objective vectors of the archive solutions are kept sorted by one of the objective values and data pertaining to the hypervolume calculation with a dimension sweep is kept for each node. For three objectives, the hypervolume contribution can be computed in $O(\mu)$ time with the HV3D$^+$-U algorithm [26] if the objective vectors of the archive solutions are stored in the data structure of the HV3D$^+$ algorithm [26]. For 4 or more objectives, we consider the use of the WFG algorithm [76], with the sorting and slicing improvements discussed in that work, which can compute the hypervolume contribution in $O(\mu^{m/2} \log \mu)$ time for m objectives.

To compute the ε-indicator of an upper bound set comprised of a single point, with respect to the archive A, we consider the problem:

$$I_\varepsilon(\{u\}, A) = \max_{1 \leq i \leq m} \max_{a \in A} a_i / u_i \tag{6.2}$$

Note that, we can maintain a list of the maximum values of a_i for each coordinate i, $1 \leq i \leq m$, in an array of size m, which is trivially updated when the archive A is updated. Then, the computation of $I_\varepsilon(\{u\}, A)$ takes $O(m)$ time. As such, calculating the binary ε-indicator is expected to be much faster than calculating the binary hypervolume for our upper bound set, which should be particularly noticeable for a high number of objectives.

If a binary quality indicator is order-preserving (or order-reversing) with respect to the weak dominance relation when fixing the first parameter, which is the case for the binary hypervolume and ε-indicator as discussed in Section 2.2, then its values can be cached for nodes in the queue that have not yet been selected and kept in a priority queue to more quickly select the next node to be processed. For a BeFS selection strategy this needs to be a global priority queue containing all the nodes that can be selected, whereas for the BeDFS selection strategy we need to maintain a set of priority queue for each level of the search tree. However, note that when considering a dichotomic branching strategy, there is in fact no need to keep any priority queue, since after selecting the most promising node on a particular level, there is only a single node remaining at that level.

One potential issue with keeping a priority queue is that when the archive maintained by the BB changes, then, depending on the quality indicator, the cached values may need to be updated and the priority queue needs to be recomputed, which can be quite costly if done often. As such, it is questionable whether or not the values should be kept in a priority queue. Our preliminary experiments suggested that for the ε-indicator it is best to maintain a priority queue at all times, since a reconstruction is rarely required. However, for the binary hypervolume indicator, we found it was best to use a priority queue only if the cached values are not updated often. In particular, we start by not considering a priority queue and instead keep all the nodes in a vector that can be searched in linear time to find the most promising node. Then, if there are 10 consecutive calls to the selection methodology that do not update the hypervolume of the archive, we build a priority queue from that vector. Once the hypervolume of the archive changes, we go back to not using a priority queue until there are 10 consecutive calls to the selection methodology that do not update the hypervolume of the archive. Finally, to avoid unnecessary quality indicator computations, we only recompute the quality indicator of the upper bound after the archive changes on the nodes for which the previously cached value was better than the current best, since the new quality indicator value will remain the same or decrease.

6.1.3 Experimental Study

In this section, we carry out an experimental study to analyze the (anytime) performance of the BeFS and BeDFS selection strategies on the MOBKP for problem instances generated according to the procedure described in Chapter 4. In particular, we considered 100 instances for each number of objectives $m \in \{2, 3, 5, 7\}$. The remaining parameters were randomly sampled from the following uniform distributions:

- $n \in \mathcal{U}\,(50, 150)$;
- $\rho_v \in \mathcal{U}\left(\frac{6}{\pi}\sin^{-1}\frac{1}{2(1-m)}, 1\right)$;
- $\rho_w \in \mathcal{U}\,(-1, 1)$;
- $\omega \in \mathcal{U}\,(0.3, 0.7)$.

We consider the following alternatives for the implementation of the BB and IBB approaches:

- **Branching**: Dichotomic;
- **Branching order**: Default (Random) or O^{sum};
- **Selection strategy**: DFS, BFS, ε-indicator BeFS, ε-indicator BeDFS, binary hypervolume BeFS, binary hypervolume BeDFS.

The BB and IBB approaches were implemented in `C++`, and the code is available at [33]. The code was compiled with `GCC 10.2.0`, and the experiments were carried out in parallel, one per thread, on a machine with two `Intel(R) Xeon(R) Silver 4210R` processors. The algorithms were run with a timeout of 100 seconds and an 8Gb memory limit.

In the results that follow we consider the performance of the BB and IBB approaches for different test scenarios. In particular, we measure the hypervolume of the archive found by an BB or IBB approach for a given problem instance considering a given CPU timeout value. There are a total of 100 instances for each number of objectives $m \in \{2, 3, 5, 7\}$, and for each instance we consider 200 distinct CPU timeout values evenly distributed on a log scale in the range $[1e{-}3, 1e2]$. Therefore, we consider a total of 20 000 test scenarios for each value of m. The code to reproduce the experiments, and to generate the tables and figures presented in the following sections, is available at [32].

2 objectives

Table 6.1 gives the ratio of test scenarios with 2 objectives for which the BB and IBB approaches using particular selection strategies found the best known archive, with respect to the hypervolume quality indicator, and considering two distinct branching orders. Note that, the best known archive is defined with respect to all the archives found by any BB or IBB approach using the same branching order for that test scenario, i.e., for that instance and CPU timeout value.

For the default branching order, we see that the IBB approaches guided by the binary hypervolume indicator have the best ratio, and that the traditional BB approaches have the worst ratio. The IBB approaches guided by the ε-indicator are slightly better than the BB approaches, but have small ratios compared against approaches guided by the hypervolume. Note that the sum of the ratios for each branching order may be greater than 1 since multiple algorithms can find an archive with the same hypervolume value. This is likely to happen in scenarios with large CPU timeout values since multiple approaches may find the Pareto set within that time.

For the O^{sum} branching order, we see that the BB approach using the DFS selection strategy has the best ratio by a large margin. This indicates that a

Order	BFS	DFS	HV-BeFS	HV-BeDFS	ε-BeFS	ε-BeDFS
Default	0.13	0.12	**0.56**	0.42	0.21	0.16
O^{sum}	0.14	**0.80**	0.28	0.21	0.18	0.18

Table 6.1: Ratio of test scenarios with 2 objectives for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

good ordering of the items for the MOBKP can lead to a good archive, in terms of hypervolume, using the DFS selection strategy. Despite this, we see that the IBB approaches, especially those guided by the hypervolume, can sometimes find better, or equally good, archives. Moreover, it should be noted that such an ordering may not be easy to achieve for every problem, and further analysis of the data reveals that the IBB approaches considering a default branching order, in particular those guided by the hypervolume quality indicator, can often achieve archives with hypervolume values that are not much worse, and sometimes even better, than those found by the DFS selection strategy with the O^{sum} branching order. This suggests that our IBB approaches are particularly interesting when a good branching order for the DFS selection strategy is not known.

Figure 6.2 gives the ratio of test scenarios with specific CPU timeout values for which a given approach was able to find the best archive. We see that for the default branching order, the IBB approach guided by the binary hypervolume using a BeDFS selection strategy got the highest ratio for small CPU timeout values. However, for larger CPU timeout values the IBB approach guided by the binary hypervolume using a BeFS selection strategy surpasses it. Note that, since we are measuring archive quality in terms of hypervolume, it is not surprising that the approaches guided by the binary hypervolume have the best performance. Also, note that the sum of ratios at CPU timeout values closer to 100 are clearly greater than for values closer to 1. We recall that this happens because for some instances the BB and IBB approaches can all find the Pareto set. For the O^{sum} branching order, we see that the BB approach following a DFS selection strategy, has the best ratio for all CPU-timeout values.

Figure 6.3 gives the mean error, and standard deviation, in terms of the difference between the hypervolume found by a given selection strategy and the hypervolume found by any other strategy considering the same branching order, when that difference is greater than 0. We discard values where the difference is equal to 0 because we want to evaluate how close a given strategy is to the best, when that strategy itself is not the best. For the default branching order, we see that the BFS and DFS selection strategies give the worst error for

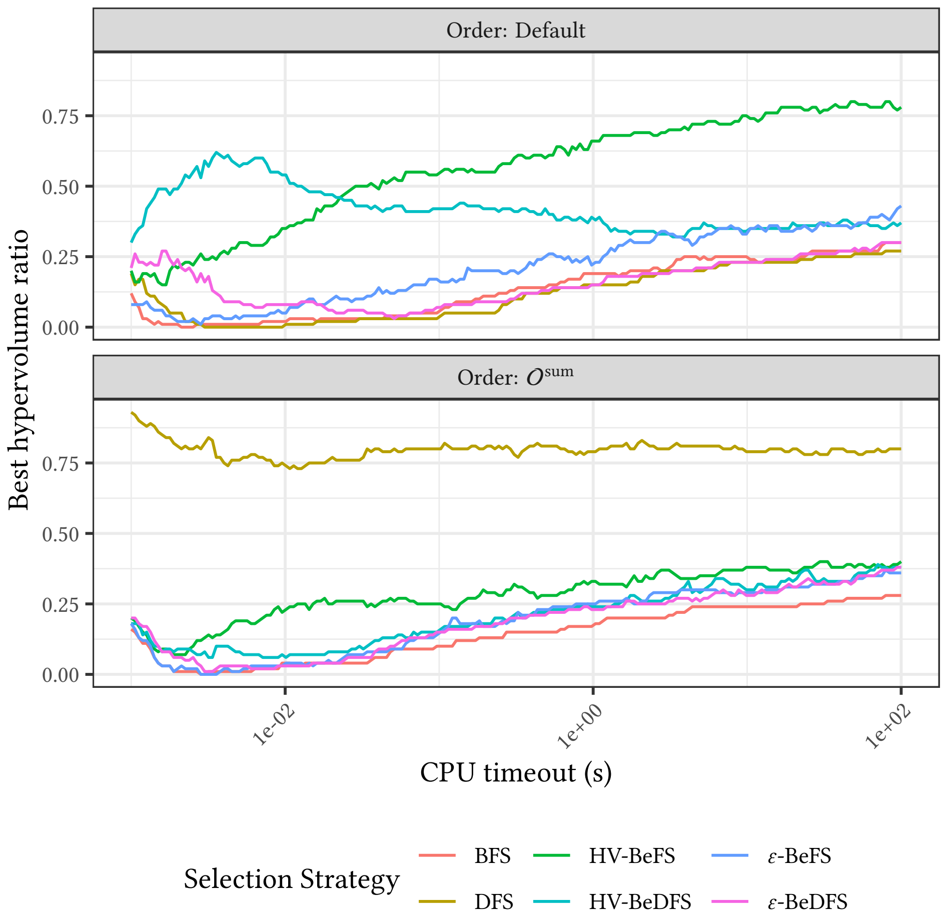


Figure 6.2: Ratio of test scenarios, with 2 objectives and a given CPU timeout value, for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

all timeout values, which means that not only can these selection strategies rarely find the best archive, as evidenced by the ratios shown previously, but also that the archives found by those strategies are significantly worse than those found by the IBB approaches. On the other hand, the BeDFS strategies give the smallest error for small timeout values, and BeFS strategies give the smallest error for large timeout values. This indicates that, even when these strategies cannot find the best archive, the difference in terms of hypervolume to the best is small. In particular, note that the approaches guided by the ε-indicator, despite having a small ratio, generally have small errors. As such all IBB approaches are generally quite good, as they often find the best archive or otherwise have a small error.

For the O^{sum} branching order, we see that the BFS strategy gives the largest error, but that the DFS strategy gives the smallest error. This is not too surprising given the results in terms of ratio. But we also see that the errors of

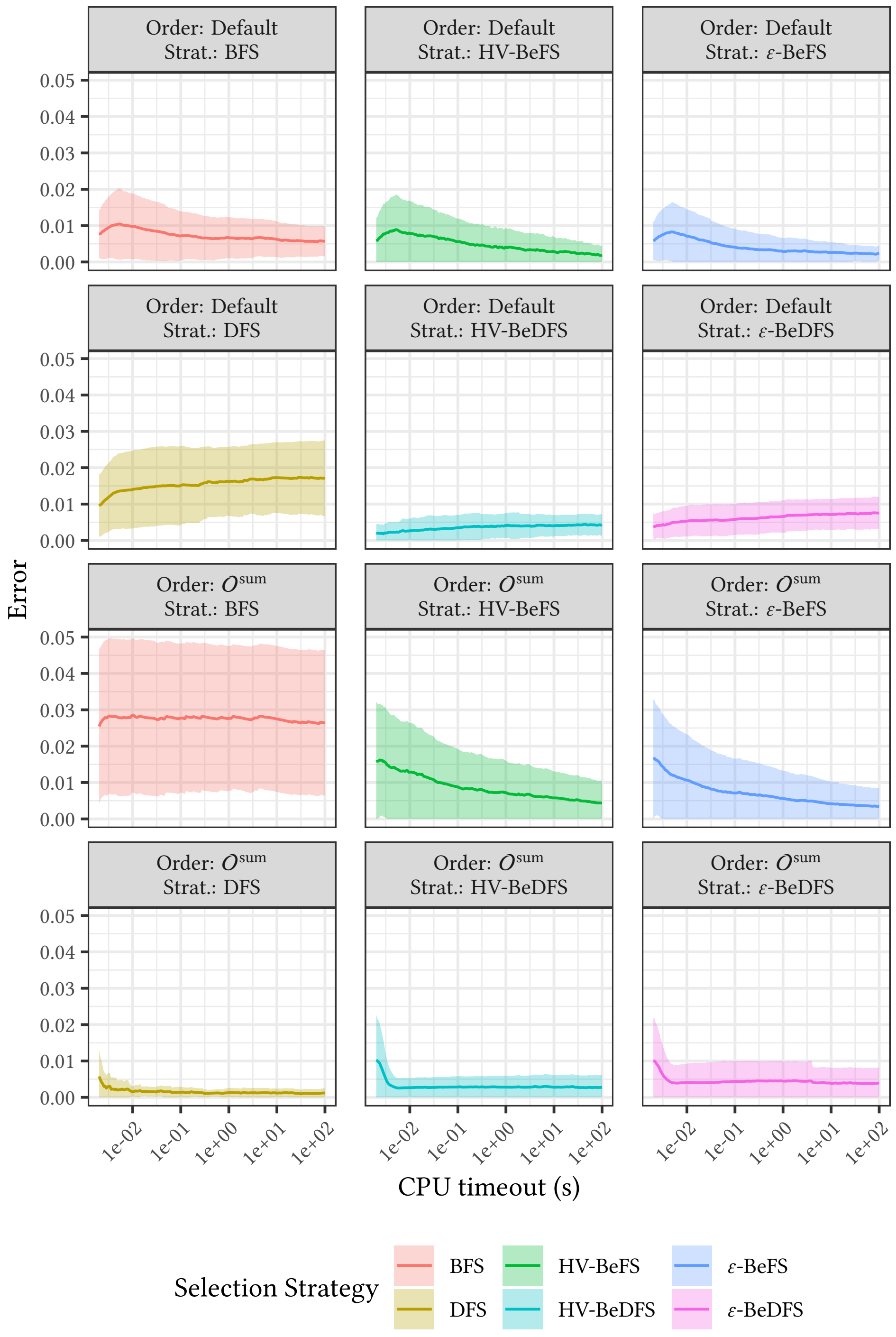


Figure 6.3: Mean and standard deviation of the difference between the quality of the archive found by a selection strategy for instances with 2 objectives and a particular CPU timeout value, and the quality of the best archive found, when that difference is greater than 0.

the IBB approaches are now more significant, with the exception of the BeDFS approach guided by the hypervolume, which generally has a small error. However, we see that the error for the BeFS strategies is decreasing significantly, and we would expect the error to become smaller than that of the BeDFS strategies for larger timeout values.

3 objectives

Table 6.2 gives the ratio of test scenarios with 3 objectives for which a given BB or IBB approach found a archive with the best hypervolume. For the default branching order, we see that the BeFS approach guided by the ε-indicator has the best ratio, followed by the IBB approaches guided by the hypervolume indicator. This contrasts with the previous scenarios with 2 objectives, where the approaches guided by the hypervolume indicator had much better ratio than the IBB approaches guided by the ε-indicator. This is not too surprising since the cost of computing the hypervolume contribution for the upper bound point is significantly higher than the cost of computing the ε-indicator contribution for instances with 3 or more objectives, as discussed in Section 6.1.2. Nonetheless, the IBB approaches guided by the hypervolume still have a higher ratio than the BeDFS approach guided by the ε-indicator, despite their increased computational cost. Lastly, we see that all IBB approaches have better ratio values than the traditional BB approaches, as in the previous case.

For the O^{sum} branching order, we see that the BB approach using a DFS selection strategy once again has the best ratio by a large margin. However, we see that this ratio decreased slightly compared to the previous case with 2 objectives. Moreover, we remark that the IBB approaches guided by the hypervolume indicator also decreased compared to the previous case, and that the BeFS approach guided by the ε-indicator increased, such that it is now the second best approach tied with the BeFS approach guided by the hypervolume indicator.

Figure 6.4 gives the ratio of test scenarios with a given CPU timeout value, for which a given approach was able to find the archive with the best hypervolume. For the default branching order, we see that the BeDFS approach guided

Order	BFS	DFS	HV-BeFS	HV-BeDFS	ε-BeFS	ε-BeDFS
Default	0.12	0.11	0.36	0.33	**0.42**	0.15
O^{sum}	0.12	**0.74**	0.23	0.15	0.23	0.15

Table 6.2: Ratio of test scenarios with 3 objectives for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

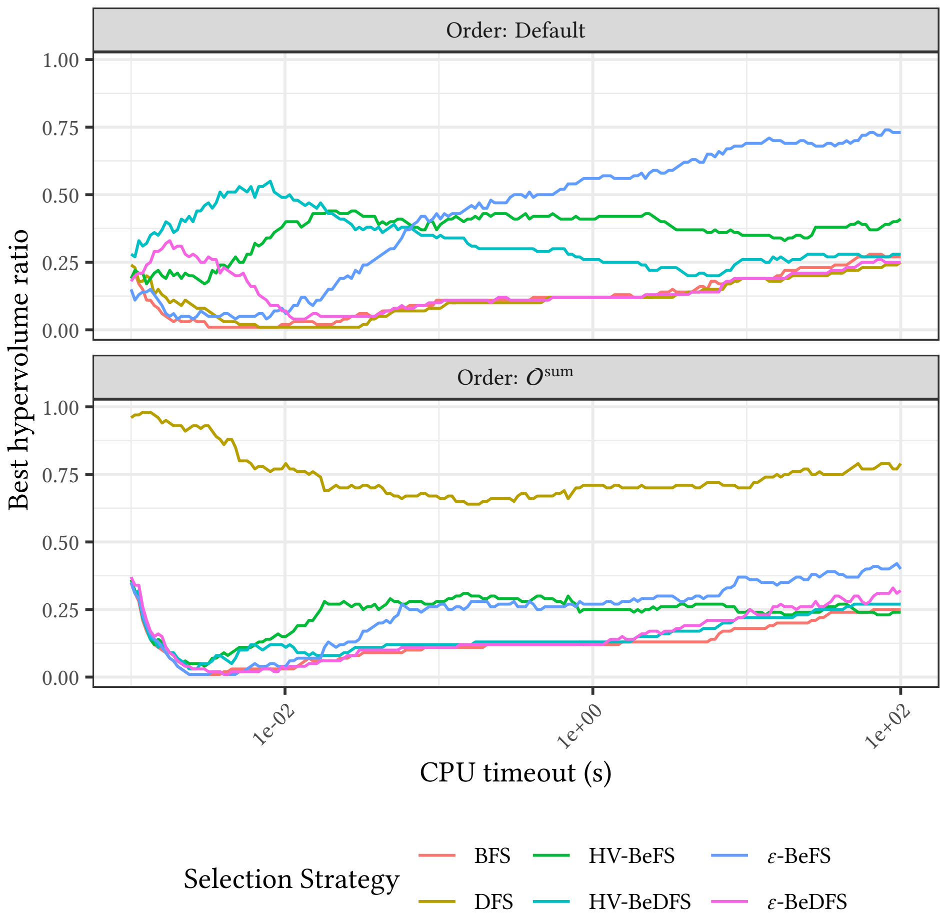


Figure 6.4: Ratio of test scenarios, with 3 objectives and a given CPU timeout value, for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

by the hypervolume indicator is the approach with the highest ratio for CPU timeout values below 1e−2. Then, it is tied with the BeFS approach guided by the hypervolume indicator until a CPU timeout value close to 1e−1. After that, the BeFS approach guided by the ε-indicator has the best ratio. This shows, that the IBB approaches guided by the hypervolume indicator are still preferable in an initial phase, despite the higher computational costs. However, for large CPU timeout values the BeFS approach guided by the ε-indicator is clearly preferred.

For the O^{sum} branching order, the DFS is clearly the best selection strategy at all times, much like it was for 2 objectives, but we see that its ratio is generally not as high as in that case. With regards to the other approaches, we see that the BeFS selection strategy guided by the hypervolume has a better ratio for CPU timeout values below 1e0, but the BeFS selection strategy guided by the ε-indicator has a better ratio for CPU timeout values greater than those.

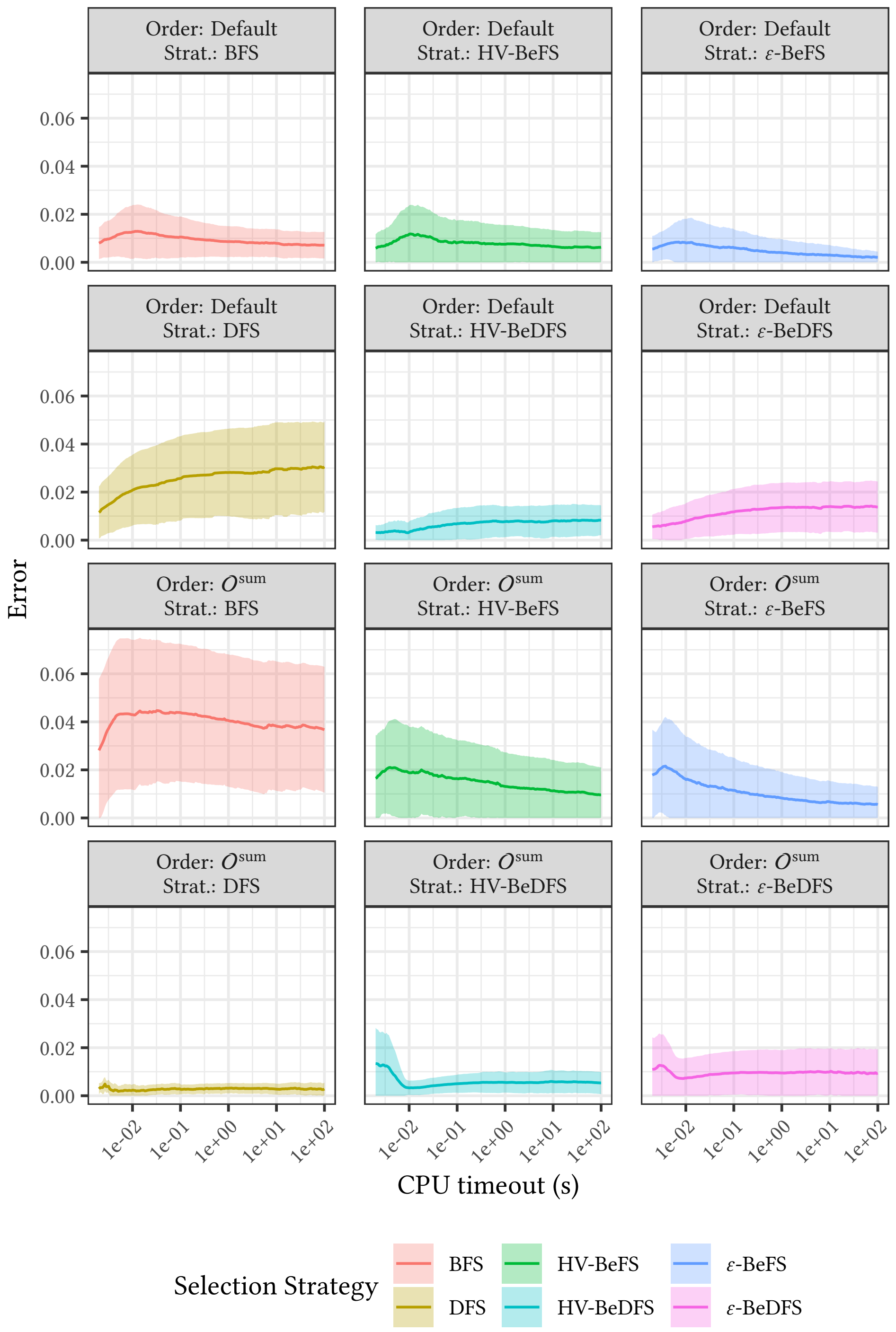


Figure 6.5: Mean and standard deviation of the difference between the quality of the archive found by a selection strategy for instances with 3 objectives and a particular CPU timeout value, and the quality of the best archive found, when that difference is greater than 0.

Figure 6.5 gives the error of the archive found by a selection strategy that did not find the best archive for instances with 3 objectives. For the default branching order, we see that the BeDFS selection strategy guided by the hypervolume indicator gives the smallest error for small timeout values, whereas the BeFS strategy guided by the ε-indicator gives the smallest error for large timeout values, which is in accordance with the ratio results. When considering the O^{sum} branching order we see that, besides the DFS strategy, these two strategies also have the smallest error.

5 objectives

Table 6.3 gives the ratio of test scenarios with 5 objectives for which the different selection strategies found the archive with the best hypervolume. For the default branching order, the BeFS selection strategy guided by the ε-indicator has the best ratio by a large margin. The second and third best ratios are for the BeDFS approaches guided by the ε-indicator and hypervolume indicator respectively. The worst approach is the BeFS approach guided by the hypervolume, which is worse than the naive BB approaches. These results indicate that the computational cost of the binary hypervolume indicator for 5 objectives is a problem for the performance of the IBB approaches guided by that indicator. For the O^{sum} branching order, we see that the IBB approaches guided by the hypervolume are also the worst. But we see that the BeFS strategy guided by the ε-indicator has a ratio that is much better than in previous cases, and not far from the ratio of the DFS strategy, which is still the best.

Figure 6.6 gives the ratio of test scenarios with a given CPU timeout value for which a given approach was able to find the archive with the best hypervolume. For the default branching order, we see that the BeDFS approach guided by the ε-indicator is the approach with the highest ratio for CPU timeout values below 2e−2. For CPU timeout values greater than 2e−2, the BeFS approach guided by the ε-indicator has the best ratio by a large margin. It is also worth noting that the IBB approaches guided by the hypervolume indicator have small ratio for almost all CPU timeout values, with the exception of the BeDFS approach which can achieve a ratio greater than 0.25 for CPU timeout values

Order	BFS	DFS	HV-BeFS	HV-BeDFS	ε-BeFS	ε-BeDFS
Default	0.10	0.09	0.04	0.14	**0.66**	0.23
O^{sum}	0.10	**0.63**	0.05	0.06	0.44	0.12

Table 6.3: Ratio of test scenarios with 5 objectives for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

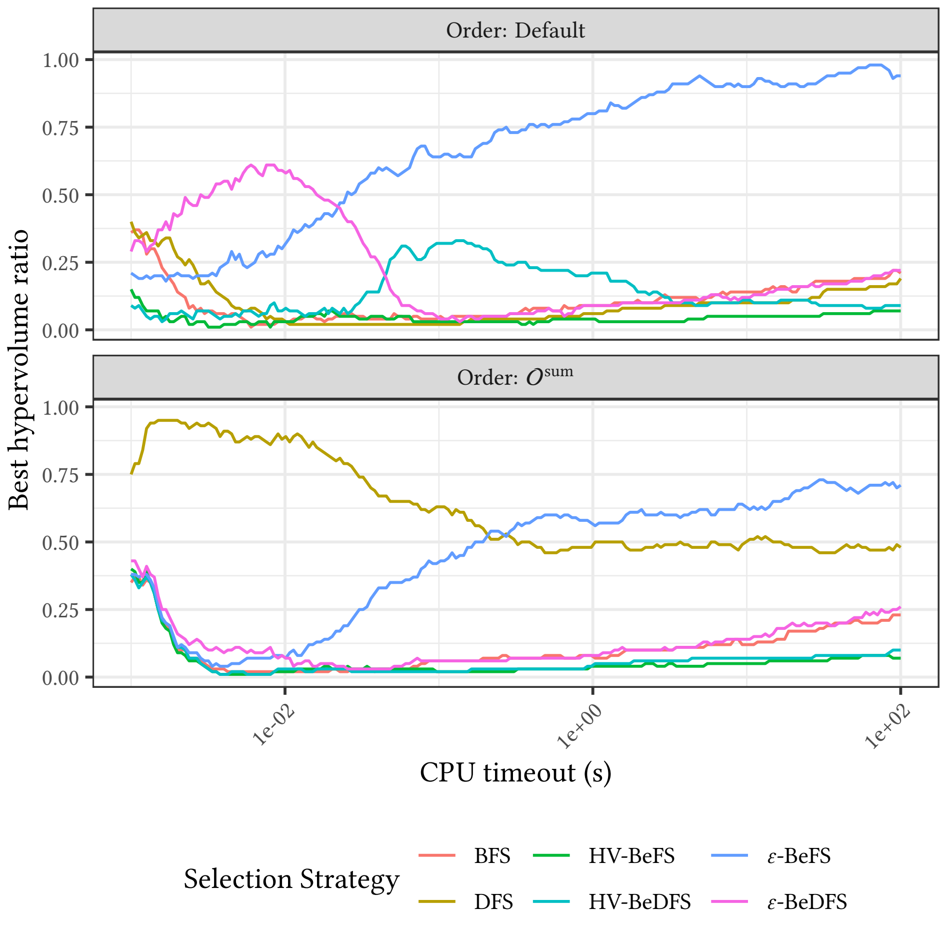


Figure 6.6: Ratio of test scenarios, with 5 objectives and a given CPU timeout value, for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

close to 1e−1, despite the fact that the binary hypervolume indicator has a high computational cost. Note that, in the BeDFS case the binary indicator does not need to be computed as often, which explains why the approach can sometimes achieve good results. Nonetheless, this shows that, as expected, the high computation cost of the hypervolume indicator has a big impact on the performance of the IBB approaches guided by that indicator. However, we see that the ε-indicator, which has a small computational cost, can achieve very good results despite the fact that we are measuring archive quality in terms of the hypervolume indicator.

For the O^{sum} branching order, the DFS is the best selection strategy for CPU timeout values smaller than 2e−1, as in the previous cases. However, we see that the BeFS approach guided by the ε-indicator has a better ratio for CPU timeout values greater than that. This contrasts with the previous cases for 2 and 3 objectives. One plausible reasoning is that it is more difficult to get a good

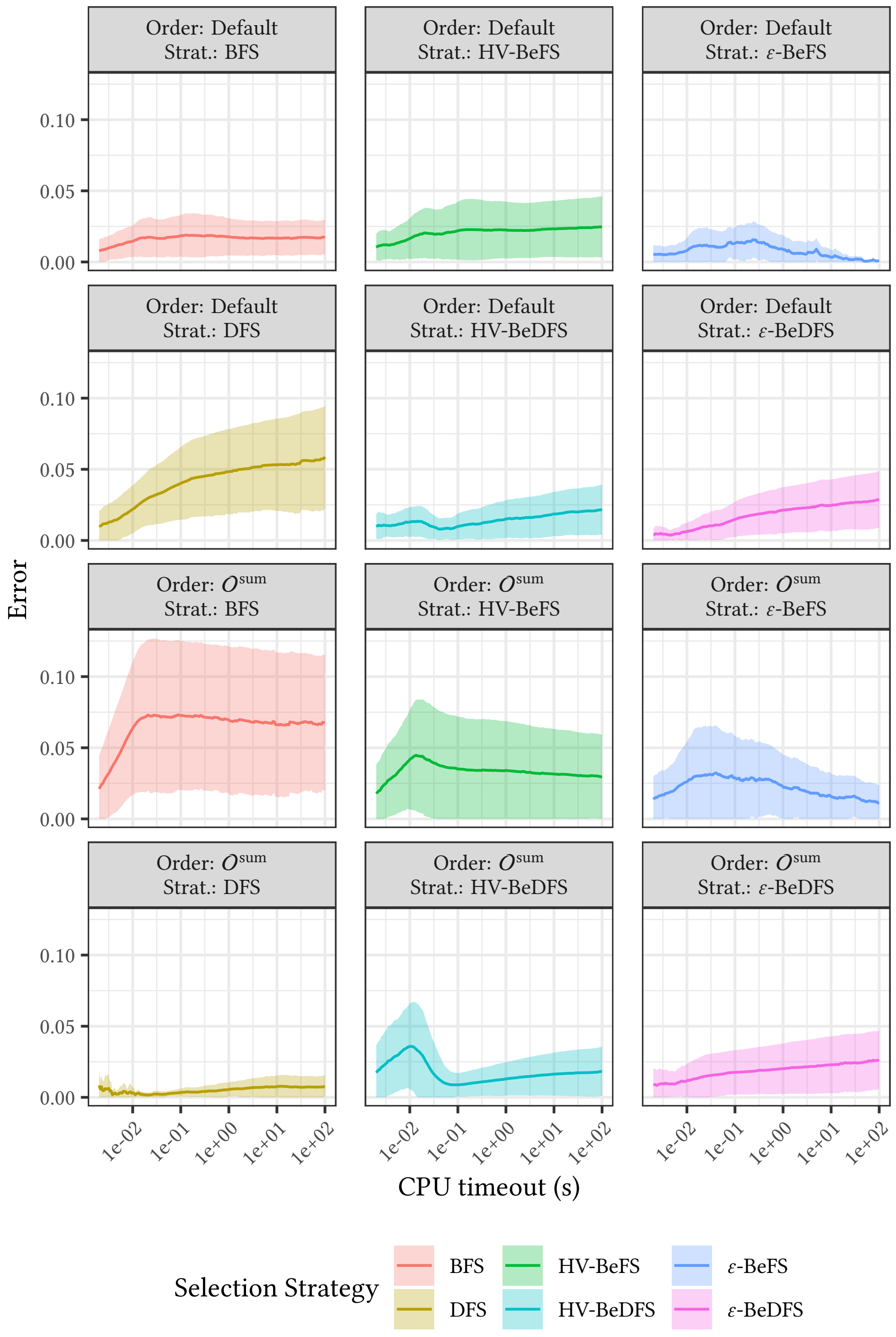


Figure 6.7: Mean and standard deviation of the difference between the quality of the archive found by a selection strategy for instances with 5 objectives and a particular CPU timeout value, and the quality of the best archive found, when that difference is greater than 0.

approximation of the Pareto front, i.e., an approximation that uniformly covers the whole Pareto front, with a static order as the number of objectives increases, due to the conflicting relation between multiple objective functions. That is, the DFS strategy can quickly find good solutions in a particular direction in objective space as it goes down in the search tree, which explains why it is initially quite good. However, the approach then struggles to find solutions that are far from that initial direction since it gets stuck at the bottom of the search tree in a particular direction if the lower and upper bounds are not very tight. On the other hand, the BeFS approach can explore different directions more easily if the indicator that guides the search favors uniformly distributed points in the objective space. Nonetheless, further analysis is needed to understand what problem properties lead to this behavior.

Figure 6.7 gives the mean and standard deviation of the error of a selection strategy for instances with 5 objectives, when it failed to find the best archive. For the default branching order, we see that the BeFS selection strategy guided by the ε-indicator most often has the smallest error, and that the BeDFS approaches guided by either indicator also have small errors when the CPU timeout values are small. These results are in line with those in Figure 6.6 since the best approaches at each stage, are also the ones that have the smallest error. For the O^{sum} branching order, we see that the errors of the IBB approaches follow a similar trend but are significantly larger. On the other hand, the DFS approach has a very small error at all times. Notably, the small error for large CPU timeout values indicates that, despite not being the approach that most often finds the best archive for such timeout values, it can very often find an archive that is close to the best.

7 objectives

Table 6.4 gives the ratio of test scenarios with 7 objectives for which the different selection strategies found the archive with the best hypervolume. The results are quite similar to those we found for 5 objectives. In particular, for the default branching order, the BeFS selection strategy guided by the ε-indicator has the best ratio by a large margin, the second and third best ratios are for the BeDFS approaches guided by the ε-indicator and hypervolume indicator respectively, and the worst approach is by the BeFS approach guided by the hypervolume. For the O^{sum} branching order, we see that the DFS strategy is the best, and the BeFS guided by the ε-indicator is the second best, whereas the other strategies are all much worse than these two. These results, again highlight that the computational cost of the hypervolume has a large impact on the performance of the IBB approaches guided by binary hypervolume.

Order	BFS	DFS	HV-BeFS	HV-BeDFS	ε-BeFS	ε-BeDFS
Default	0.07	0.08	0.01	0.12	**0.62**	0.21
O^{sum}	0.06	**0.65**	0.03	0.04	0.39	0.09

Table 6.4: Ratio of test scenarios with 7 objectives for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

Figure 6.8 gives the ratio of test scenarios with a given CPU timeout value for which a given approach was able to find the archive with the best hypervolume. Once again, the results are similar to those for instances with 5 objectives. In particular, for the default branching order, the BeDFS approach guided by the ε-indicator has a large ratio for small CPU timeout values, and the BeFS approach guided by the ε-indicator has a much larger ratio than all others

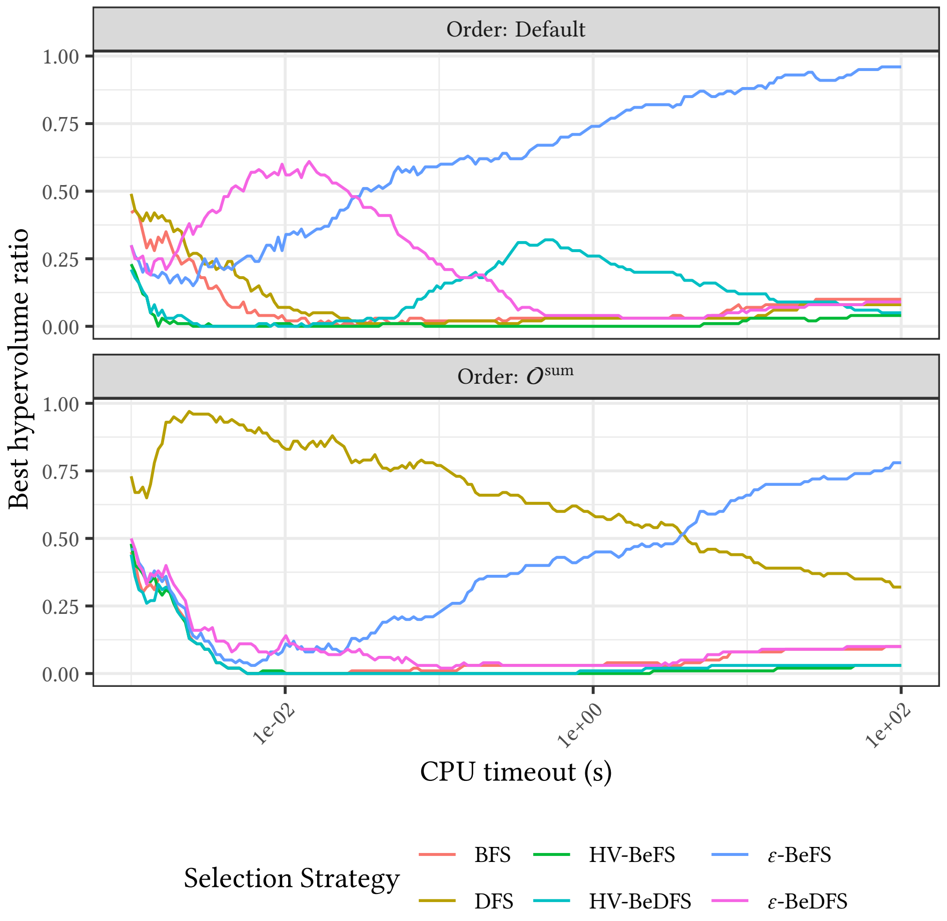


Figure 6.8: Ratio of test scenarios, with 7 objectives and a given CPU timeout value, for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

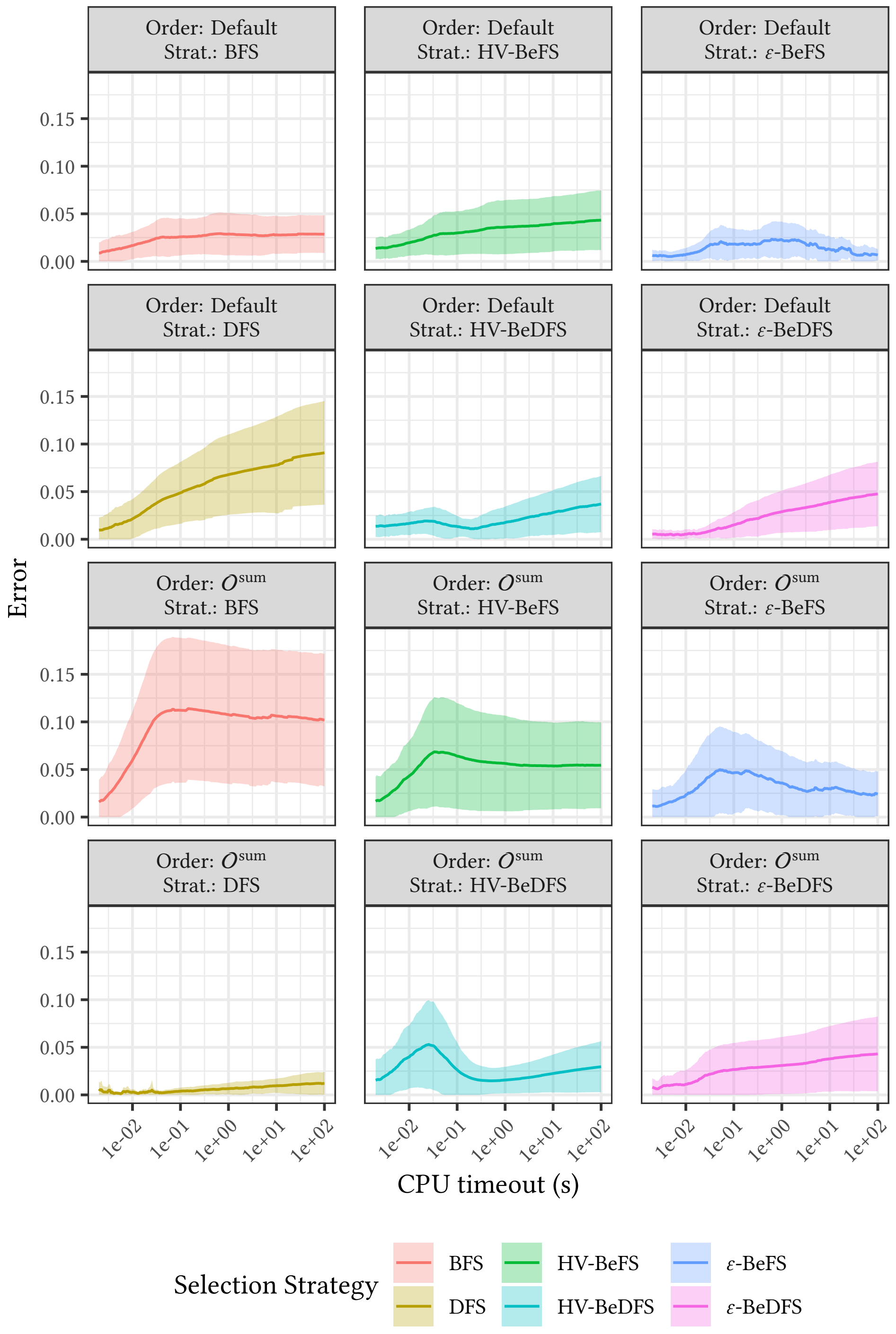


Figure 6.9: Mean and standard deviation of the difference between the quality of the archive found by a selection strategy for instances with 7 objectives and a particular CPU timeout value, and the quality of the best archive found, when that difference is greater than 0.

strategies for large CPU timeout values. For the O^{sum} branching order, we see that the DFS strategy has the best ratio most often, but that for large CPU timeout values the BeFS approach guided by the ε-indicator is better. Notably, the difference between the ratio of this strategy and the ratio of the DFS strategy is much greater than for 5 objectives, which suggests that as the number of objectives increases the difference is likely to be more significant.

Figure 6.9 gives the mean error and corresponding standard deviation for each selection strategies, when the archive found for instances with 7 objectives was not the best. The results are similar to those for instances with 5 objectives. However, we can see that both the errors are slightly larger in this case, which indicates that the error increases with the number of objectives.

6.1.4 Discussion

In the previous section we compared the proposed IBB approaches guided by the binary hypervolume and ε-indicator against the naive BB approaches using a DFS and BFS search strategies to solve instances of the MOBKP. In particular, we analyzed which approaches found the best archive with respect to the hypervolume quality indicator on a set of instances and CPU timeout values, considering two distinct branching orders. We also analyzed the mean, and standard deviation, of the difference in relative hypervolume between the archive found by a given approach, and the best archive found by any strategy for the same CPU-time.

For a default branching order, which corresponds to a random ordering of the items, we saw that the IBB approaches often found the best archive, and gave the smallest error. In particular, the BeDFS selection strategies often had the best performance for small CPU timeout values, and the BeFS strategies had the best performance for medium and large CPU timeout values. Moreover, for 2 objectives the IBB approaches guided by the hypervolume performed better, and for 5 and 7 objectives the IBB approaches guided by the ε-indicator performed better. For 3 objectives, the BeDFS guided by the hypervolume had better performance for small CPU timeout values, and the BeFS guided by the ε-indicator had better performance for large CPU timeout values. As such, we conclude that for these MOBKP instances, and for a random branching order our IBB approach is significantly better than the naive DFS and BFS selection strategies.

For the O^{sum} branching order the results favored the DFS strategy in most scenarios, which indicates that a good branching order is particularly important for that strategy. However, it is worth noting that such an order might not be available for every problem. Moreover, we saw that for large CPU timeout values and instances with 5 and 7 objectives, the DFS approach started to have worse performance in terms of finding the best archive, and the BeFS strategy guided by the ε-indicator was more often able to find the best archive. Still, the difference in archive quality between the archive found by the DFS strategy and the best archive was quite small, which means that the archive found was quite good despite not being the best.

Further analysis also revealed that the difference in archive quality between the best archive found by the IBB approaches with the default order, and the best archive with the O^{sum} branching order, was often quite small. This suggests that the IBB approaches can often find good archives without the need to devise any particular branching order.

For future work, we consider that it would be relevant to study the performance of the IBB approaches on a broader set of instances to better understand what characteristics of the problem are significant to each strategy. One question that arose, but for which preliminary analysis revealed no answer was for what kinds of instances is the DFS strategy with a O^{sum} branching order not the best approach. Moreover, it would be relevant to analyze the impact of different lower and upper bound sets on the performance of the IBB approaches. Lastly, it would be interesting to analyze the performance of the IBB approaches on different problems, in particular, problems for which no clear branching order is known.

6.2 Branch-and-Bound Online Selection

6.2.1 Methodology

In Section 6.1.3, we highlighted that IBB approaches following a BeDFS search strategy can achieve good performance for small CPU timeout values, but that for large medium and large CPU timeout values the BeFS search strategies more often had the best performance. Analyzing the anytime traces of the approaches also revealed that the BeDFS can often find an archive with good quality, but then struggles to significantly improve the archive over time. By contrast, BeFS approaches cannot find such a good archive for small CPU times, but can consistently improve archive quality over time.

Taking these observations into account, we propose an online selection methodology to select between BB search strategies in order to improve anytime performance. We start with an IBB approach that uses a BeDFS search strategy. Then, we swap to a BeFS search strategy once we consider that the initial approach can no longer improve archive quality significantly. In particular, we monitor the relative difference in archive quality, with respect to the total archive quality, over the last k explored nodes. If this relative difference is below a certain threshold value, denoted by δ, then we swap approaches. Note that, following the analysis in Section 6.1.3, a BeDFS search strategy guided by the binary hypervolume is initially considered if $m < 4$, or guided by the ε-indicator if $m \geq 4$. Then, we swap to a BeFS search strategy guided by the binary hypervolume if $m < 3$, or guided by the ε-indicator if $m \geq 3$.

Finally, we consider swapping back to the original BeDFS search strategy if the number of active nodes is greater than a parameter ℓ, to avoid running out of memory.

6.2.2 Experimental Study

In this section, we carry out an experimental study to analyze the (anytime) performance of the online selection methodology described in the previous section. We empirically set the parameters $k = 100$, $\delta = 1\text{e}{-6}$, and $\ell = 100000$ for the online selection methodology since these parameters gave good results in preliminary experiments. For comparison, we consider the same instances and algorithms used for the experiments in Section 6.1.3.

The online methodology was implemented in `C++`, and the code is available at [33]. The experiments were carried out in parallel, one per thread, on a machine with two `Intel(R) Xeon(R) Silver 4210R` processors. The algorithms were run with a timeout of 100 seconds and an 8Gb memory limit. We consider the performance of the BB, IBB, and online approaches on the same scenarios presented in Section 6.1.3. The code to reproduce the experiments and to generate the tables and figures presented in the following sections is available at [32].

2 Objectives

Table 6.5 gives the ratio of test scenarios with 2 objectives for which an approach found the best archive. For the default branching order, we see that the online selection methodology has a high ratio (0.83) that is significantly larger than all other approaches. This indicates that our approach is working correctly, and that the changing algorithms positively improves anytime behavior. For the O^{sum} branching order, we see that the DFS approach still has the best ratio, which is not surprising since it was the best approach in the previous experiments, and was not considered for the online selection methodology. However, it is worth noting that the ratio of the DFS approach decreased slightly, from 0.8 to 0.72, compared to the ratio values without the online selec-

Order	BFS	DFS	HV-BeFS	HV-BeDFS	ε-BeFS	ε-BeDFS	Online
Default	0.12	0.12	0.28	0.16	0.13	0.14	**0.83**
O^{sum}	0.14	**0.72**	0.20	0.20	0.17	0.18	0.34

Table 6.5: Ratio of test scenarios with 2 objectives for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

tion methodology, which means that our online methodology approach could sometimes find a better archive in cases where the individual IBB approaches could not. Nonetheless, these results indicate that taking into account the ordering of the items to select the initial selection strategy could improve the performance of our online selection methodology further. Lastly, we see that, as for the default order, the online selection methodology has the best ratio of all other approaches.

Figure 6.10 gives the ratio of test scenarios for which a given approach was able to find the best archive, at different CPU timeout values. For the default branching order, we see that the online selection methodology has the best ratio at all times. Note that, despite the fact that our online selection methodology starts with the BeDFS selection strategy guided by the hypervolume, that same approach is still the second best for small CPU timeout values despite both approaches being deterministic. This is due to the fact that we are considering anytime performance in terms of CPU time, as such, slightly different conditions during the execution of the experiments can lead to either approach having better performance. However, we see that for CPU timeout values close to 1e−2, when the BeFS selection strategy guided by the hypervolume surpasses the BeDFS, the online methodology is significantly better than the latter, which indicates that the online methodology is in fact changing to the BeFS approach correctly, and that this increases anytime performance significantly. For the O^{sum} branching order, we see that the online selection methodology is often better than the IBB approaches.

Figure 6.11 gives the mean error, and standard deviation, in terms of the difference between the relative hypervolume of an archive found by a given selection strategy and the best archive found by any other strategy considering the same branching order, when that difference is greater than 0. We see that, for both branching orders, the online approach has a very small error. As such, we see that the online approach is not only good in terms of the ratio, but also that when it does not find the archive with best quality for a particular test scenario, it finds an archive that is very close to the best.

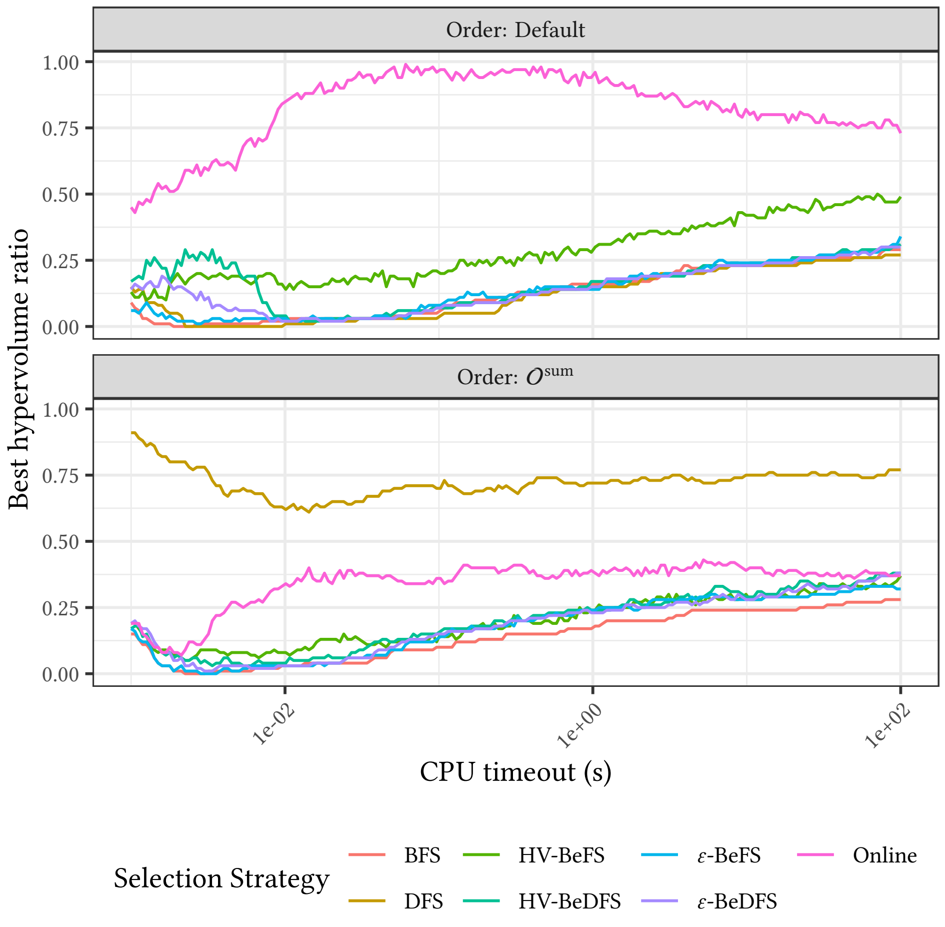


Figure 6.10: Ratio of test scenarios, with 2 objectives and a given CPU timeout value, for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

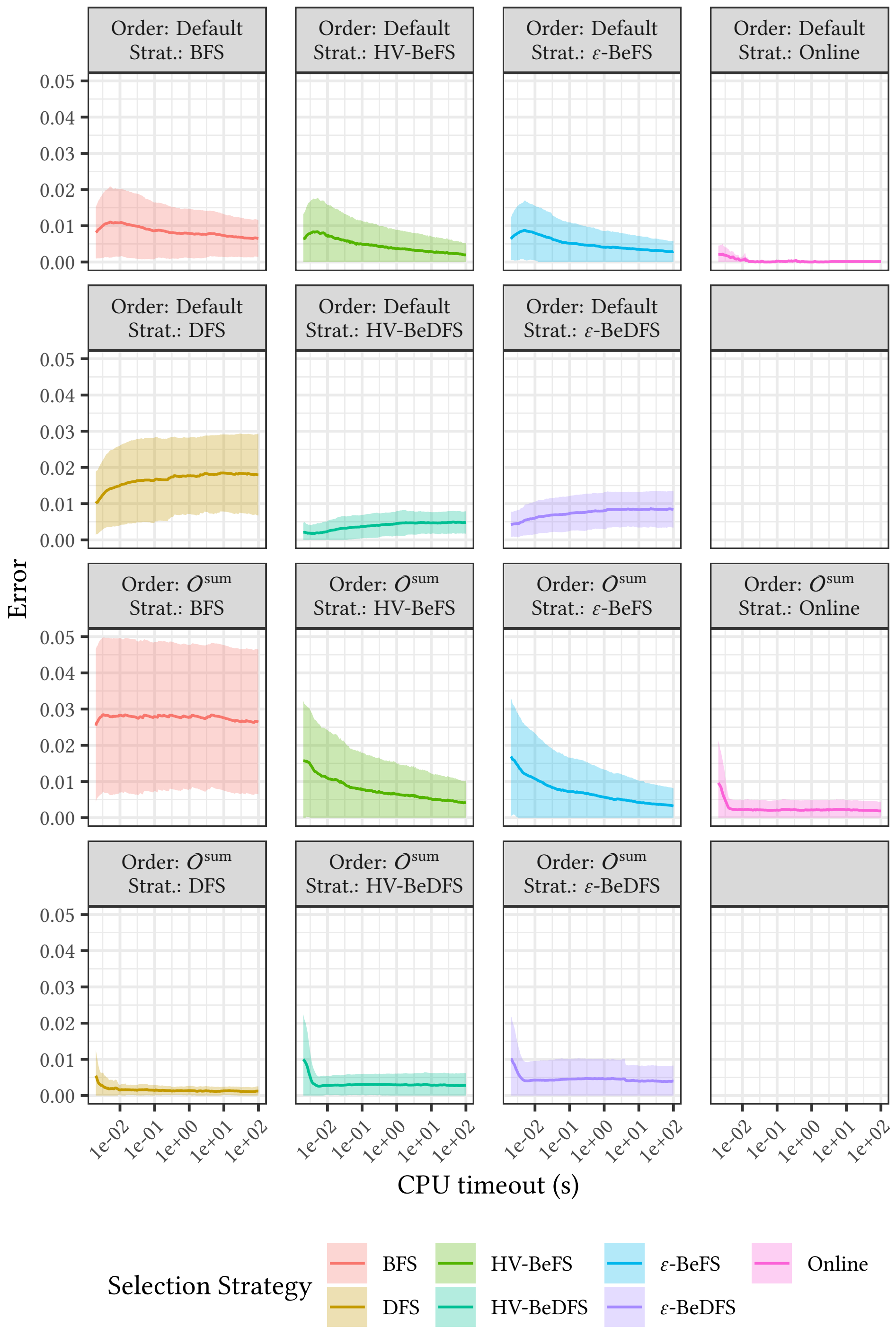


Figure 6.11: Mean and standard deviation of the difference, in terms of hypervolume, between the archive found by a selection strategy and the best archive found, when the difference is greater than 0.

3 Objectives

Table 6.6 gives the ratio of test scenarios where each approach found the best archive. For the default branching order, we see that the online selection methodology still has the best ratio. However, there is a decrease compared to the previous case with 2 objectives. Moreover, we see that the BeFS strategy guided by the hypervolume increased slightly, and that the other approaches have similar ratio values. Compared to the results with 3 objectives, but without the online selection methodology, we see that both the strategies considered in our online selection methodology, that is, the BeDFS strategy guided by the hypervolume and the BeFS strategy guided by the ε-indicator, decreased significantly, while the other two IBB strategies did not. As the BeFS strategy guided by the hypervolume has a ratio that is quite significant, this suggests that opting not to consider this strategy in our approach is detrimental to the performance of our online selection methodology. As such, to further improve our online methodology we should consider both BeFS strategies when changing algorithms. Unfortunately, preliminary experiments did not reveal any relation between the characteristics of the instances and which BeFS strategy had the best performance. Therefore, further analysis is required to better understand this, and successfully select between the two strategies.

For the O^{sum} branching order, the results are similar to the previous case with 2 objectives. In particular, we see that the online selection methodology had better performance than the other IBB approaches, and that the ratio values for the DFS strategy decreased slightly, from 0.74 to 0.7, compared to the results without the online selection methodology.

Figure 6.12 gives the ratio at the sampled CPU timeout values. For the default branching order, we see that, while in the previous case with 2 objectives the online methodology got a ratio value close to 1 for certain CPU timeout values, in this case the maximum ratio value is close to 0.75. The fact that the BeFS strategy guided by the hypervolume has significant ratio values, indicates that not considering that strategy for online selection is decreasing the performance of our online methodology. Still, the online methodology has

Order	BFS	DFS	HV-BeFS	HV-BeDFS	ε-BeFS	ε-BeDFS	Online
Default	0.11	0.11	0.34	0.15	0.12	0.14	**0.65**
O^{sum}	0.12	**0.70**	0.21	0.14	0.14	0.15	0.30

Table 6.6: Ratio of test scenarios with 3 objectives for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

the best ratio at all times, which shows that it is expectedly improving any-time performance. For the O^{sum} branching order, we see that the DFS strategy clearly has the best ratio at all CPU timeout values. Still, we see that, as with 2 objectives, the online selection methodology is often the best between the other approaches.

Figure 6.13 gives the error for test scenarios with 3 objectives, at the sampled CPU timeout values. As with 2 objectives, we see that the online selection methodology has a very small error for both branching orders. This means that even if it does not find the archive with the best quality, it can find an archive with quality close to that.

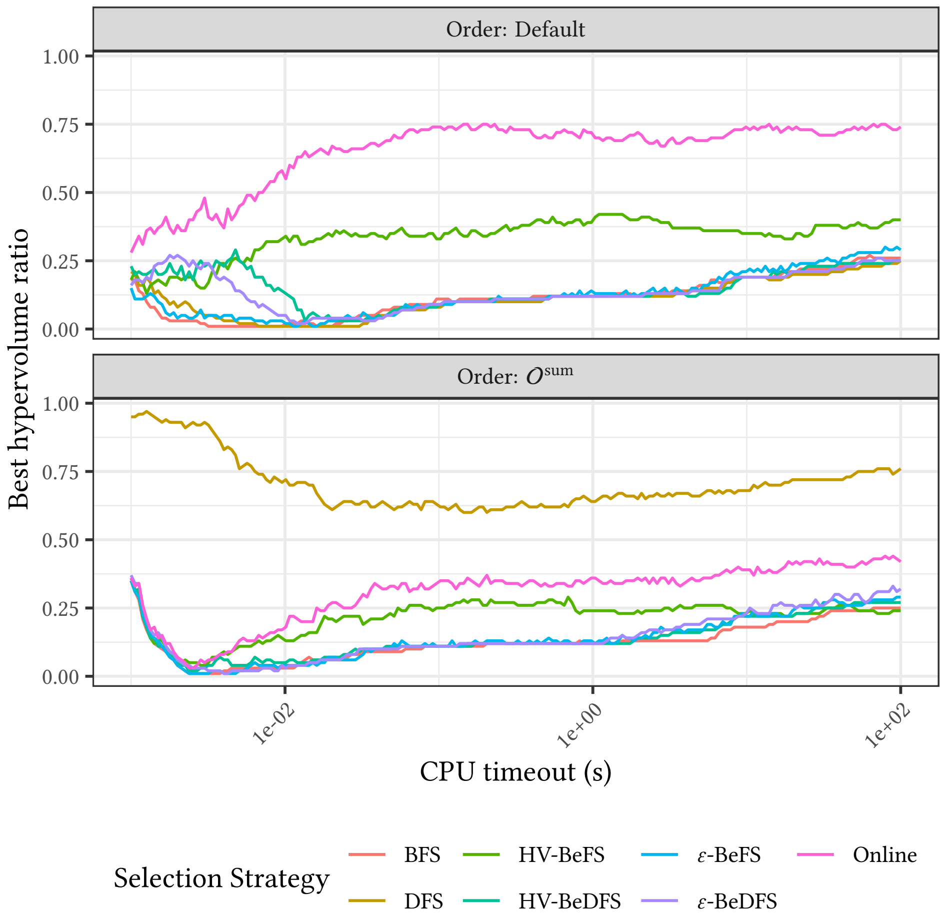


Figure 6.12: Ratio of test scenarios, with 3 objectives and a given CPU timeout value, for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

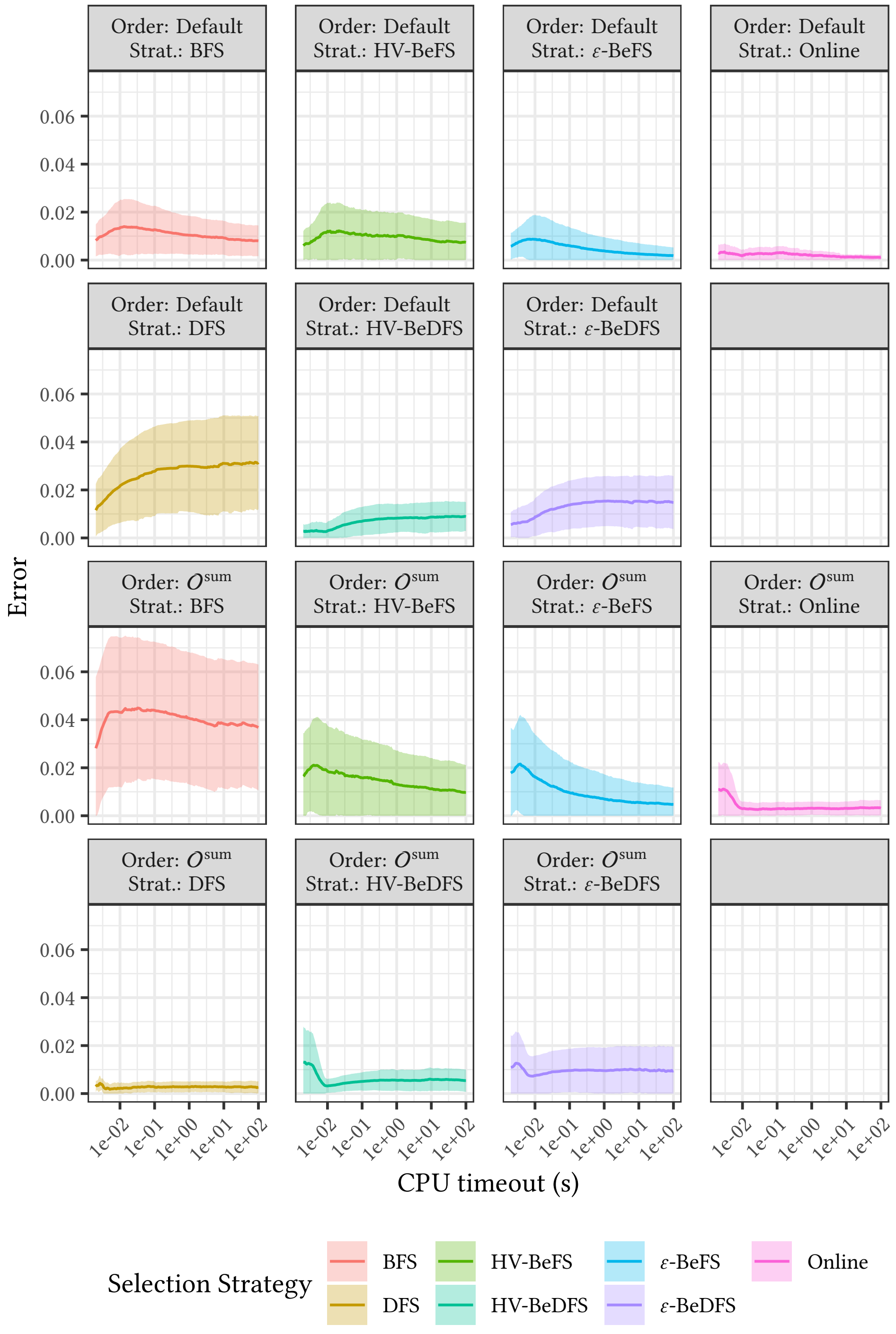


Figure 6.13: Mean and standard deviation of the difference, in terms of hypervolume, between the archive found by a selection strategy and the best archive found, when the difference is greater than 0.

5 Objectives

Table 6.7 gives the ratio of test scenarios with 5 objectives for which each approach found the best archive. We see that the online selection methodology has the best ratio for the default branching order, as in previous cases. However, in contrast with previous cases, the BeFS strategy guided by the ε-indicator has the best performance among all others. This is not too surprising given the results we saw without the online selection methodology. However, it is somewhat surprising that the value is so high considering that we are using that strategy in our online approach. Moreover, the value for the online selection methodology is not as high as in previous cases. For the O^{sum} branching order, we see that, as in previous cases the DFS strategy has the highest ratio, but that its ratio is slightly lower than for the experiments without the online selection methodology. However, more surprisingly, we see that the online selection methodology is the best among the other approaches, but the BeFS strategy guided by the ε-indicator has a ratio value that is only slightly worse.

Order	BFS	DFS	HV-BeFS	HV-BeDFS	ε-BeFS	ε-BeDFS	Online
Default	0.10	0.09	0.04	0.09	0.30	0.14	**0.60**
O^{sum}	0.10	**0.59**	0.05	0.06	0.22	0.11	0.36

Table 6.7: Ratio of test scenarios with 5 objectives for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

Looking at the ratio over different CPU timeout values in Figure 6.14, we see that at CPU timeout values close to 1e−2 the online selection methodology improves significantly, which is to be expected due to the swap between strategies, and indicates that the methodology is correctly, and favorably, changing strategies. However, at around 1e−1, we see a meaningful drop in the ratio values of the online selection methodology. This appears to coincide, first with an increase in ratio for the BeDFS strategy guided by the hypervolume indicator, and second with an increase in ratio from the BeFS strategy guided by the ε-indicator. Analysis of the individual anytime traces shows that the BeDFS strategy guided by the hypervolume can sometimes achieve very good archive quality at such CPU times, despite a generally worse start compared to the same strategy guided by the ε-indicator. Therefore, if a DM establishes that small CPU timeout values are not relevant, and we incorporate that preference into the selection, then it could make sense to choose the BeDFS guided by the hypervolume initially. For the second case, the anytime traces reveal that the BeFS strategy guided by ε-indicator has an anytime performance that is very

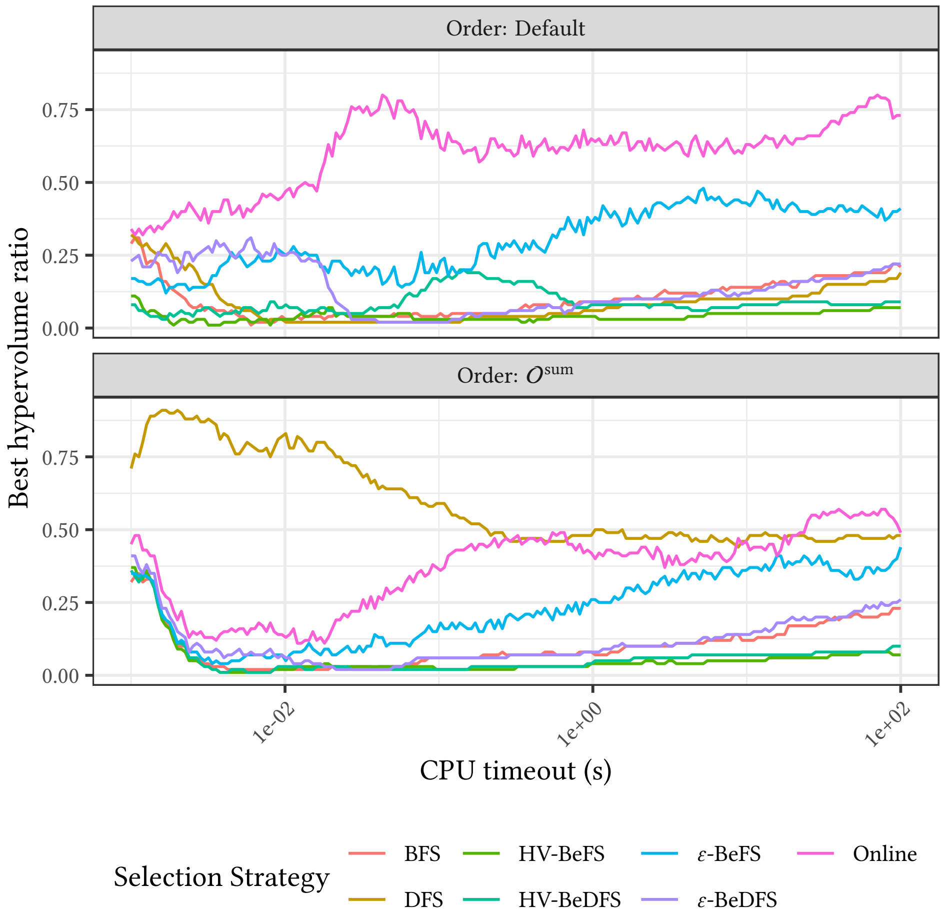


Figure 6.14: Ratio of test scenarios, with 5 objectives and a given CPU timeout value, for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

similar to that of the online methodology for large CPU time values. Therefore, the difference in archive quality between the two is very small, and the drop in ratio is, for that reason, not very significant.

For the O^{sum} branching order, we see that the DFS approach has very good ratio values in the beginning but at around 1e−1 our online methodology has similar ratio values. Moreover, for large CPU timeout values the BeFS approach guided by the ε-indicator surpasses both the online selection methodology and the DFS. As discussed in the previous paragraph, this happens because for large CPU time values both the online methodology and the BeFS strategy have very similar anytime performance. Therefore, the difference in ratio between the two is not significant.

Figure 6.15 gives the error for each strategy, when it could not find the archive with the best quality, in terms of the difference between the quality of the archive it found and the best. For the default branching order, we see that,

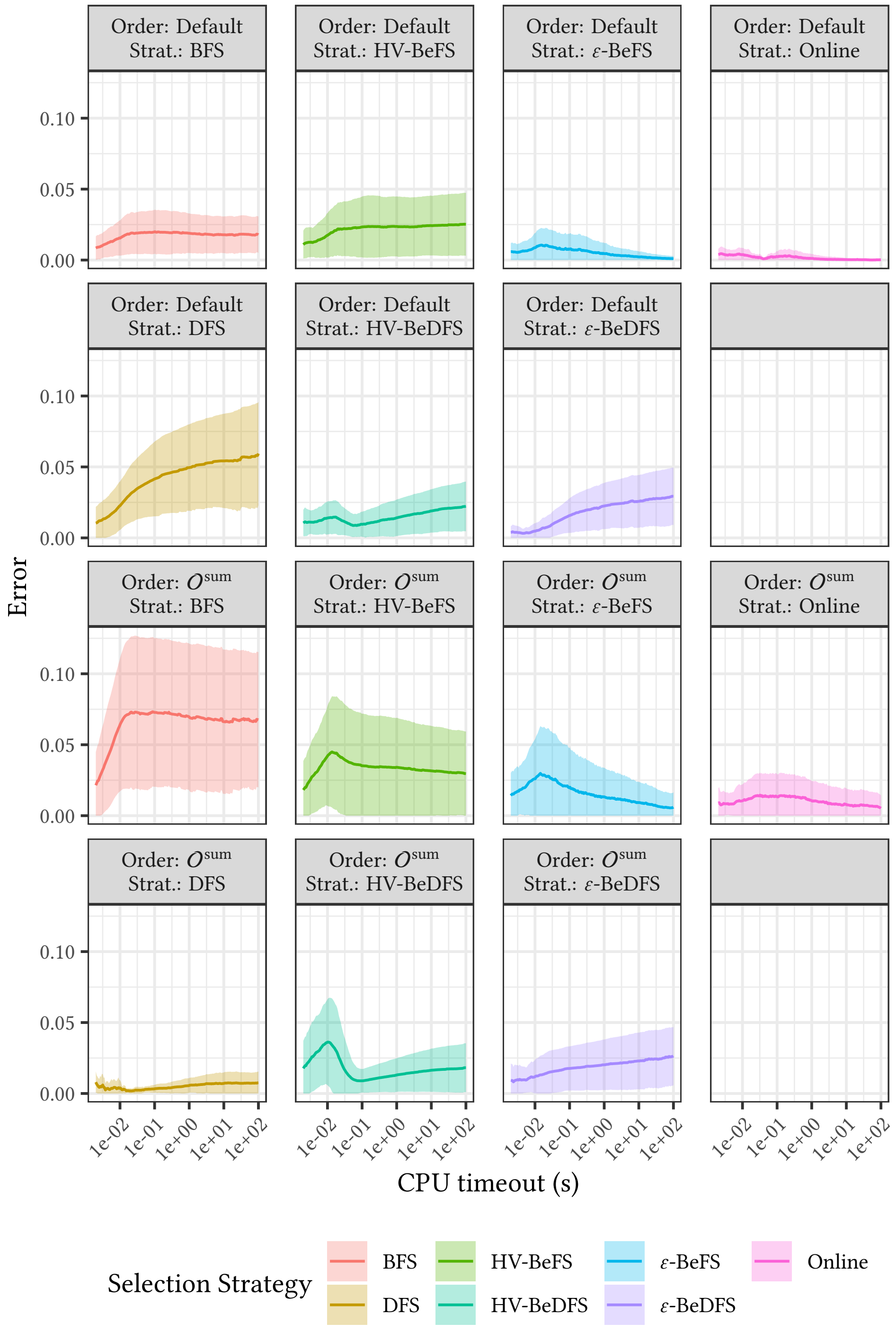


Figure 6.15: Mean and standard deviation of the difference, in terms of hypervolume, between the archive found by a selection strategy and the best archive found, when the difference is greater than 0.

as in the previous cases, the error of the online selection methodology is very small for all CPU timeout values. On the other hand, we see an increase in its error for the O^{sum} branching order compared to the previous cases. Nonetheless, it is still the IBB approach with the smallest error.

7 Objectives

Table 6.8 gives the ratio of test scenarios with 7 objectives for which each strategy was able to find the archive with best quality. Overall, we see that results are similar to the case of 5 objectives. In particular, for the default branching order the online methodology has the best ratio value, and the BeFS strategy guided by the ε-indicator the second best. For the O^{sum} branching order, we have that the DFS has the best ratio, followed by the online selection methodology and the BeFS strategy guided by the ε-indicator.

Order	BFS	DFS	HV-BeFS	HV-BeDFS	ε-BeFS	ε-BeDFS	Online
Default	0.07	0.07	0.01	0.08	0.26	0.10	**0.58**
O^{sum}	0.06	**0.58**	0.03	0.03	0.19	0.08	0.35

Table 6.8: Ratio of test scenarios with 7 objectives for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

Figure 6.16 gives the ratio values over the CPU timeout values. For the default branching order, we see that the online selection methodology is not always the best for very small CPU timeout values. However, analysis of the individual anytime traces reveals that this happens because both the online methodology and the BeDFS strategy guided by the ε-indicator have very similar anytime behavior in the beginning, since the former is using the latter, and due to fluctuations runtime conditions the latter is sometimes better. However, after CPU timeout values 1e−2, we see that the online selection methodology increases significantly due to a meaningful change of algorithms. Then, at CPU timeout values 1e−1 we see a slight decrease. As in the previous case with 5 objectives, this is first due to the BeDFS strategy guided by the hypervolume indicator sometimes having better performance than the IBB approaches guided by the ε-indicator at those times, and later because the BeFS strategy guided by the ε-indicator has very similar anytime performance compared to the online selection methodology which uses the former. For the O^{sum} branching order, the results are very similar to the case with 5 objectives.

Figure 6.17 gives the error of each approach at the sampled CPU timeout values. For the default branching order, we see that the online methodology has

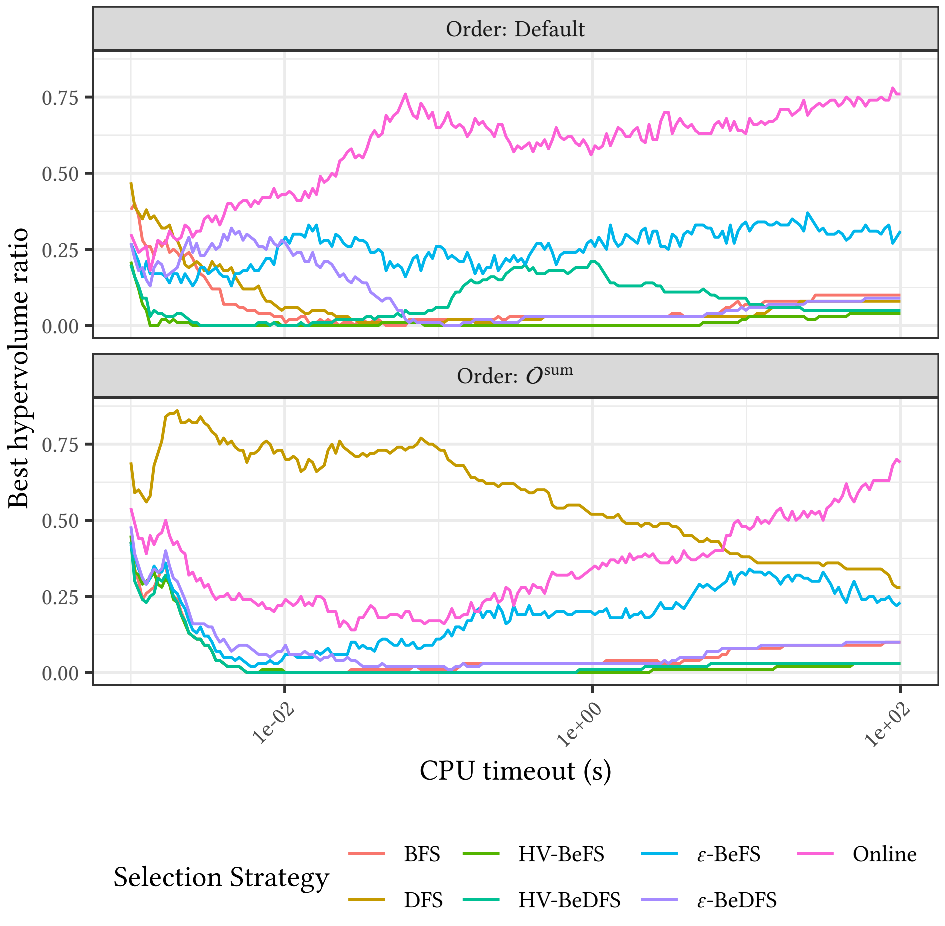


Figure 6.16: Ratio of test scenarios, with 7 objectives and a given CPU timeout value, for which a selection strategy found the best archive with respect to the hypervolume quality indicator.

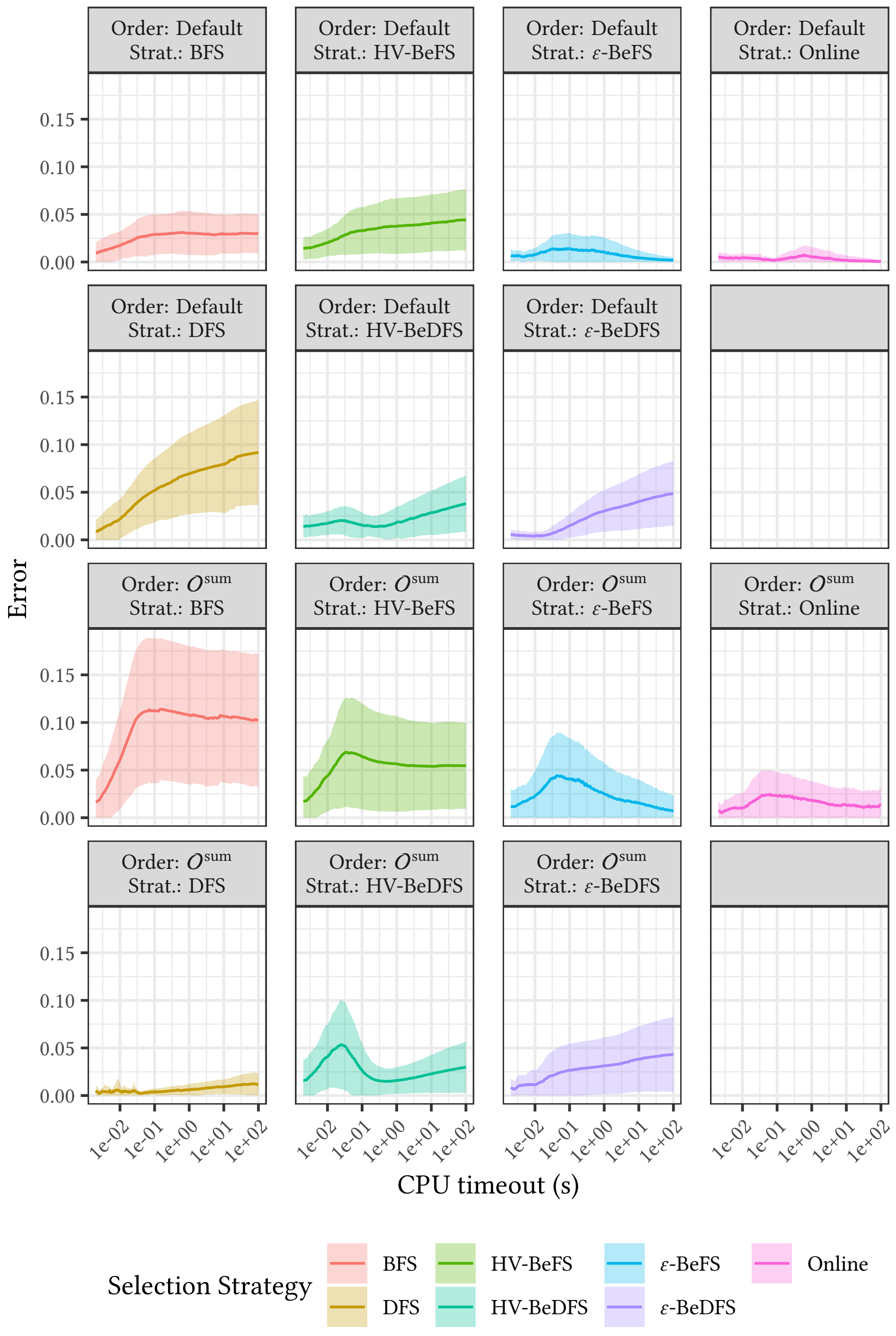


Figure 6.17: Mean and standard deviation of the difference, in terms of hypervolume, between the archive found by a selection strategy and the best archive found, when the difference is greater than 0.

the smallest error at all times, as in the previous cases. For the $\mathcal{O}^{\text{sum}}$ branching order, we see that, besides the DFS approach, the online methodology generally has the smallest error. However, at CPU timeout values close to 1e−1, the BeDFS approach guided by the hypervolume has a smaller error, which is in line, with what was explained in the previous paragraph with regards to the anytime performance of the approach for those CPU time values.

6.2.3 Discussion

The results for the online selection methodology presented in the previous section show that this methodology does improve anytime performance, as we expected during its conception. In particular, we see that the ratio of test scenarios for which this methodology was able to find the archive the best quality is generally very high when considering a default branching order. For the $\mathcal{O}^{\text{sum}}$ branching order, we saw that the DFS strategy is still often the best. However, there were a small number of scenarios where the DFS strategy was better than the individual IBB, but worse than the online methodology. Nonetheless, this means that only considering IBB approaches is not always the best approach, since with a good branching order the DFS strategy often found the best archive.

We also saw that fixing the initial and secondary strategies a priori resulted in some cases where the other IBB approaches that were not chosen were able to find an archive with better quality. As such, in the future, it would be relevant to consider an online selection methodology that is capable of selecting which approaches are the best taking into account the characteristics of the instance being solved, and the anytime preferences of the DM, similarly to the offline selection methodology presented in Chapter 5.

Lastly, in this experimental study we chose parameters k, δ and ℓ that gave good results during a preliminary analysis but did not thoroughly study their effect. As such, it would be relevant to investigate the impact of these parameters in the future.

6.3 Conclusion

In this chapter, we started by presenting the IBB framework, which consists of a BB for multi-objective optimization problems guided by quality indicators. Then, we performed an experimental study on the MOBKP considering four distinct IBB approaches, that use the BeFS and BeDFS selection strategies, and two quality indicators, the binary hypervolume and the ε-indicator, and two naive BB approaches that use the DFS and BFS selection strategies. The experimental results showed that our IBB approaches are very often better than the naive BB approaches, when considering a random ordering of the items for the MOBKP. Note that we say that an approach is better than another for a given test scenario, if the quality of the archive returned by that approach, in terms of hypervolume, is better. Moreover, we noted that the approaches guided by the hypervolume indicator were often better than the ones guided by the ε-indicator for instances with 2 objectives, which is to be expected since we are measuring quality in terms of the hypervolume indicator. However, with the increase in the number of objectives, the approaches guided by ε-indicator became better due to the computational overhead of the hypervolume. We also noted that a good ordering of items can lead to very good results for the DFS strategy. Despite this, we saw that the difference in quality between the archive found by some IBB approaches, and the archive found by the DFS strategy, was often not very large.

Then, we considered an online selection methodology that can change between IBB strategies online, while solving a previously unseen problem instance. This is of particular interest, since in the previous results we had seen that BeDFS strategies were often quite good for small CPU time values, whereas BeFS strategies were better for large values. Therefore, our online selection methodology changes between these two strategies. The empirical results showed that changing algorithms did in fact result in significant improvements in anytime performance, and this approach was most often the best for a random ordering of the items. However, we also noted that, by only considering two IBB approaches for each problem meant that the results were not as good as they could have been. Therefore, one direction for future work is to consider an online selection that can select between any number of BB and IBB approaches, and perhaps even other algorithms, taking into account the characteristics of the problem instance. Moreover, we also did not take into account the anytime preferences of the DM during the selection. As such, another direction for the future is to incorporate these anytime preferences into the online selection.

Finally, it would also be relevant to study the performance of the IBB framework, and of the online selection methodology, on other problems, and to compare it against other state-of-the-art algorithms.

Chapter 7

Conclusion

7.1 Concluding Remarks

In this thesis, we considered the problem of automatically selecting between algorithms for MOO problems. In particular, we focused on the selection between anytime MOO algorithms, such that the anytime performance of the chosen algorithm is optimal with respect to the preferences of the DM. However, before looking at the ASP problem itself, we started by considering the problem of theoretically and empirically modeling the anytime performance of MOO algorithms. In the following, we discuss the main contributions of each chapter, first related to the theoretical and empirical models of anytime performance for MOO algorithms, and then with regards to the offline and online selection of MOO algorithms.

7.1.1 Theoretical Models

In Chapter 3, we presented two variants of a theoretical model of anytime performance for scalarization techniques to MOO problems with 2 objectives that find, at each iteration, an efficient solution that maximizes the hypervolume contribution. It is worth noting that, when developing these models, there was no known scalarization technique that found efficient solutions that maximized hypervolume contribution. However, Paquete et al. [61] have recently proposed such a scalarization technique, whose anytime behavior can, by definition, be modeled by these theoretical models.

The first model is given by an analytical formulation that, while simple and computationally fast, requires several assumptions that may not always be practical, such as the shape of the non-dominated set and the location of the reference point for the hypervolume indicator. The second model is given by an algorithm designed to overcome these limitations. It can be used for any problem where the set of non-dominated points can be approximated by a piecewise linear function with any number of linear segments, and for any reference point. One disadvantage of the second model is that it is slower to compute than the first model. Still, empirical results showed that it is very fast for parameterizations that we consider reasonable for most scenarios, and as such we expect that this is not likely to be a problem in practice.

An experimental study was carried out on two variants of the bi-objective knapsack problem, showing that these models, especially the second one, can very accurately model the behavior of a scalarization technique that finds a solution that maximizes hypervolume contribution at each iteration. Moreover, the model also proved useful in the design of a ε-constraint scalarization approach that has very good anytime performance. In particular, we used the model to set the constraint values for the scalarization problem to be solved at each iteration. This approach guided by our model, showed anytime performance that is very close to the anytime performance of a technique that finds a solution that maximizes the hypervolume contribution at each iteration.

7.1.2 Empirical Models

One disadvantage of the proposed theoretical models is that they cannot model the anytime performance in terms of CPU time, which we wished to consider for the ASP in this thesis. As a result, in Chapter 4, we proposed three frameworks for the development of empirical models that, in order to predict the anytime performance of MOO algorithms on previously unseen instances, take into account the anytime performance gathered from runs of those algorithms on known problem instances.

The first framework, which builds an anytime performance profile of an algorithm for the previously unseen problem instance from the anytime data gathered from runs on known instances, can easily be applied to any algorithm and should generally be easy to implement. However, it cannot be used to predict the anytime performance for CPU times for which anytime data was not gathered from actual runs on known problem instances. Moreover, it requires that known problem instances share similarities, in terms of anytime performance and problem instance features, to unseen problem instances. We presented an experimental study to analyze the quality of this framework by implementing an empirical model to predict the anytime behavior of three MOO algorithms for MOBKP problem instances with 2, 3, and 5 objectives. The results showed that the model could predict the anytime performance in most cases. However, for one of the algorithms, the quality of the prediction was often not very good on instances with 3 and 5 objectives. We believe this was likely due to fact that we had a small number of training instances and instance features.

The second framework considers fitting a (non)-linear model for each known anytime performance profile in order to predict anytime behavior for a previously unseen problem instance. As such, empirical models taking into account this framework should be able to predict, to a certain degree, the anytime performance for instances that are different than the ones we have for training, and for CPU-times for which no anytime data is available. However, it requires that a (non)-linear model can be fitted to the known anytime data, which might not be trivial. In particular, we performed some preliminary experiments, in which we were able to sometimes fit a non-linear growth model to the anytime traces of the PLS and BHV-DP algorithms. However, we found instances for which the same model was clearly not adequate.

The third framework is based on the idea of using a theoretical model that consider time in terms of the number of iterations, such as the theoretical models proposed in Chapter 3, and transforming it to a model that considers anytime performance in terms of CPU time by using empirically collected anytime data of how long each iteration takes. This framework may also allow to predict approximation quality for CPU time values for which there is no empirical data, contrary to the first framework. Moreover, it does not require that the anytime performance can be modeled by (non)-linear growth model. However, it requires that a theoretical model is known for the algorithm, which is admittedly not very common.

At this point, it is worth noting that although our primary interest for the development of these theoretical and empirical models was in regards to the ASP, they also have other applications. For example, these models can be used for the problem of automatically configuring the parameters of an algorithm, to monitor and compare the performance of algorithms, and to decide when to stop or restart an algorithm. Moreover, although we studied the empirical models in the context of MOO, they are applicable to predict the anytime performance single-objective optimization algorithms as well.

7.1.3 Offline Algorithm Selection

In Chapter 5, we presented an offline algorithm selection framework for selecting which anytime MOO algorithm to execute for a previously unseen instance. An important aspect in the formulated ASP is that we consider the anytime preferences of a DM with respect to what the anytime conditions for stopping the algorithm were, in terms of execution time and archive quality. To this end, we presented a measure of anytime performance that takes into account a utility function describing the utility of execution time and archive quality to the DM. Moreover, we assumed that these conditions were only known when giving an instance to the selection methodology, and not before; i.e., they were not known when training the selection methodology. With this in mind, we highlighted that traditional classification and regression approaches were not particularly suited for this selection problem.

As a result, we proposed an offline selection methodology that takes into account the models of anytime performance such as the ones presented in Chapters 3 and 4. In particular, it uses models of anytime performance to predict the anytime performance profile of each algorithm on a previously unseen instance, and then applies the anytime performance measure with the utility function defined by the DM to the predicted anytime performance profile, in order to select the best algorithm.

We carried out an experimental study for selecting between two and three algorithms to the MOBKP, using the first empirical model to predict anytime performance, and taking into account two distinct utility functions. The results showed that our algorithm selection methodology could often select the best algorithm, and was better than always selecting the same algorithm for nearly every scenario. One aspect that is particularly relevant to highlight is that our methodology could often select the best algorithm even in scenarios where the quality of the prediction of anytime performance given by the empirical model was not the ideal, as long as the general tendency of anytime behavior was captured. However, we did see a decrease in selection accuracy in such cases. This shows that the quality of selection is closely related to the quality of the model, and we expect that improving the quality of the empirical model will result in an improved quality for the selection.

7.1.4 Online Algorithm Selection

In Chapter 6, we considered the ASP in an online perspective, that is to select and change algorithms while solving a problem instance. In particular, we considered selecting between BB strategies taking into account the anytime performance of the currently selected strategy and previously gathered empirical knowledge about each strategy.

To this end, we started by developing and analyzing several BB strategies. In particular, we presented the IBB framework, that takes into account the quality of the upper bound set of each node, in terms of quality indicator such as the hypervolume and ε-indicator. Then, we performed an experimental study to compare the anytime performance of different IBB strategies, and traditional BB strategies, on the MOBKP. Experiments revealed that the IBB approaches were often much better than traditional BB approaches when no heuristic for branching is available. However, when a good heuristic for branching is available, we saw that the traditional DFS strategy was quite good. We also noted that, the best strategy often varied with time, and that depending on the number of objectives, the best quality indicator to guide the IBB approaches also changed.

With this information, we devised a simple online selection methodology that could change between IBB strategies. This online selection methodology proved to be quite effective, and was very often the best approach for a random branching order. However, we identified several scenarios where we expect that anytime performance can still be improved by considering a more complex selection strategy. In particular, we only considered changing between two strategies that were manually pre-selected before execution from the empirical results we had collected. However, there were several cases in which another approach was sometimes better. In addition, we only consider selecting between IBB strategies. However, when a good branching order was available the traditional DFS strategy was often better than IBB strategies. Therefore, if we also considered traditional strategies in specific scenarios we could improve the anytime performance of our online methodology. These situations suggest that a more automatic online methodology, that takes into account instances features and selects between all known strategies based on those features, could achieve better anytime performance.

7.2 Future Work

Throughout this thesis, we highlighted several opportunities for future work with respect to each aspect that was studied, which we summarize in the following sections. We leave out of this summary, future work related to the study of the algorithms, models, and selection methodologies on different problems and instances, since it would obviously be interesting to analyze them on different problems, especially on real-world problems. Moreover, we also leave out future work related to the comparison with other approaches, which would also be relevant. Instead, we focus on discussing aspects for the algorithms, models, and methodologies that we believe could be improved in the future, or even novel approaches based on the ideas discussed in this thesis.

7.2.1 Theoretical Models

A shortcoming of our theoretical model, and consequently of the proposed GEPS approach, is that it currently only works for MOO problems with 2 objectives. Therefore, given the positive results we obtained so far, an important direction for future work would be to extend the theoretical model for more dimensions. We consider that there are two main challenges for this. The first is that the approximation of the non-dominated set can no longer be defined by a set of linear segments. We believe that for 3 objectives, a representation given by a set of polygons is feasible, albeit harder to maintain. However, we have not yet thought about extending it for more dimensions. Another possibility, even for 2 objectives, would be to consider curved surfaces for approximating the surface. These may be harder to work with initially, but may more generally extend for more objectives. The second challenge is related to keeping the set of unexplored regions up to date, since finding a point in a region can potentially affect regions other than its own. This makes it harder to define an analytical model, and makes the algorithmic model less efficient as it can no longer fully cache the best values for each region.

Another important aspect for our theoretical model that deserves more research is how to approximate the non-dominated set. In this work, we only considered that the non-dominated set could be well approximated by a quadrant of a superellipse, which was in turn approximated by a set of linear segments. This worked quite well for the knapsack problems considered. However, it obviously does not work for every problem. One possibility would be to consider a subset of supported solutions given by a weighted sum scalarization, such as the DWS, to derive the linear segments that make up the piecewise approximation. This has an intrinsic computational cost depending on how difficult or easy it is to solve such scalarized problems. Moreover, depending on the problem, the set of supported solutions found may not give a good approximation. Other possibilities, could, for example, rely on theoretical results on the shape of the non-dominated set, or on using empirically data to estimate the piecewise linear function from problem instance features.

7.2.2 Empirical Models

In the implementation of the empirical model following the first framework, we considered a rather simple selection strategy based on the k-nearest neighbor algorithm that selects instances that have similar instance features compared to the previously unseen problem instance for which we want to predict anytime performance, since we expect that this will mean that they will have similar anytime performance compared to the previous unseen instance. This selection strategy showed very positive results for some scenarios. However, a natural direction for future research is to consider the use of different selection strategies, or to further study the parameterization of the current strategy. For example, we considered using a fixed value of k anytime traces during the selection. However, this can lead to selecting anytime traces from instances that are quite far from the instance for which we want to predict anytime performance if there is only a small number of similar instances. Instead, it would be interesting to consider dynamically setting the value of k depending on the instance for which we are attempting to predict performance. Alternatively, we could explore assigning different weights for the anytime traces depending on the distance, which would require incorporating such weights in the construction of the anytime performance profile.

Another aspect that deserves further investigation is the metric used to tune and evaluate the model. In particular, the measure of anytime performance that we considered cannot properly capture the tendency of anytime behavior, since, to put it simply, it only captures the area under an anytime trace and as such, two very distinct anytime traces may have the same measure. As such, a measure that could capture the difference in trend between the predicted and real anytime performance profile could lead to a better tuning and numerical evaluation of the model.

Lastly, another direction for future work is to implement empirical models following the second and third frameworks proposed. We did consider a preliminary model following the second framework that showed some promising results in preliminary experiments, but which had some issues. Nonetheless, we believe that if properly investigated it can lead to particularly accurate predictions.

7.2.3 Offline Algorithm Selection

The proposed offline algorithm selection methodology depends heavily on the models used to predict the anytime performance of each algorithm. Still, in this thesis we only investigated its performance for a particular case of a single empirical model. As such, the first aspect that we consider requires further work is the study of how the algorithm selection methodology relates to the chosen model. In particular, it would be interesting to study whether a theoretical relation between the quality of the model and the quality of the selection can be derived. Moreover, it would be interesting to study how changes to the model impact the quality of the selection in practice.

Another aspect that was not investigated in this thesis, but that is relevant for any algorithm selection methodology, is how long does algorithm selection take, and how does this impact the accuracy of the selection. In particular, it would be relevant to analyze the computational effort needed to perform algorithm selection, and to see how considering the selection time in the anytime performance of the algorithm impacts the selection accuracy of our methodology, since time spent selecting an algorithm is time that is not being spent executing the algorithm.

Lastly, another important direction for future work would be to consider new selection methodologies that could be compared to our own. In particular, despite the fact that we have identified potential issues with respect to the use of more traditional classification and regression approaches considering the dynamic anytime preferences of a DM, there may be more restricted scenarios where such approaches are viable and worth exploring.

7.2.4 Online Algorithm Selection

With regards to the proposed IBB approach, one clear direction for future research is to study the impact of different lower and upper bound sets on the performance of our approach, compared to more traditional BB approaches. Moreover, it would also be relevant to study the use and impact of other quality indicators.

With regards to the proposed online selection methodology, one clear shortcoming is that it currently depends on manually setting two strategies between which we can swap. Moreover, changing between strategies depends only on the anytime performance of the initial strategy, but not on the expected performance of the next strategy. As such, two important aspects to investigate further are the use of more than two strategies in the online selection methodology, and the automated selection of which strategy to use at each stage depending on its predicted anytime performance, similar to what we did for the offline algorithm selection. Another issue, is that the current online selection methodology was only considered for changing between BB approaches that use different selection strategies. Therefore, it would relevant to consider changing between algorithms other than BB variants. One difficulty that we anticipate is that the change between distinct algorithms may not be ideal, in the sense that some information collected by one algorithm may not be leveraged by the other, and some calculations may need to be computed from scratch.

Lastly, another direction for future work is to incorporate the anytime preferences of a DM into the online selection methodology as we did with the offline algorithm selection methodology, such that strategies can be selected with respect to those preferences.

Acronyms

AEPS Adaptive ε-constraint 18

ASP Algorithm Selection Problem 2, 9, 28, 30, 127, 163–166

BB Branch-and-Bound 18–20, 127, 128, 131–133, 136, 139, 145–147, 160, 161, 166, 167, 170

BeDFS Best Depth First Search 20, 128–134, 136, 139, 140, 142, 143, 145–148, 151, 154, 157, 160, 161

BeFS Best First Search 20, 128, 130–134, 136, 137, 139, 140, 142, 143, 145–148, 151, 154, 155, 157, 160, 161

BFS Breadth First Search 20, 128, 132–134, 145, 160

BHV-DP Bazgan-Hugot-Vanderpooten Dynamic Programming 23, 62, 68, 69, 76–78, 80, 85, 87, 89, 94, 96, 104, 112, 113, 115–123, 165

CCUBKP Capacity Constrained UBKP 44–46, 48–50

DFS Depth First Search 20, 128, 132–134, 136, 137, 139, 140, 142, 145–147, 151, 152, 154, 155, 157, 160, 161, 167

DM Decision Maker 1–3, 5, 25, 28, 107, 108, 110, 125, 154, 160, 161, 163, 165, 166, 170

DP Dynamic Programming 22, 23, 44

DWS Dichotomic Weighted Sum 16, 24, 168

FPSV-DP Figueira-Paquete-Simões-Vanderpooten Dynamic Programming 24

GEPS Guided ε-constraint 49, 50, 55, 62, 68, 69, 98, 104, 112–116, 167

GNLS Generalized Non-Linear Least Squares 105

IBB Indicator-based Branch-and-Bound 127, 128, 130–134, 136, 137, 139, 140, 142, 145–148, 151, 157, 160, 161, 166, 167, 170

LOESS Locally Estimated Scatterplot Smoothing 70

LOOCV Leave-One-Out Cross-Validation 67, 70, 76, 82, 85, 91, 95, 98

MAE Mean Absolute Error 67, 68, 70, 71, 76, 82, 85, 91, 95, 98

MOBKP Multi-Objective Binary Knapsack Problem 7, 15–18, 21–24, 29, 50, 57, 58, 62, 63, 65, 68, 104, 108, 111, 124, 127, 130, 131, 133, 145, 160, 161, 164, 166

MOCO Multi-Objective Combinatorial Optimization 9, 10, 15, 16

MOO Multi-Objective Optimization 1–5, 7–10, 12, 16, 20, 25, 26, 28–31, 33, 43, 107, 128, 163–165, 167

PLS Pareto Local Search 24, 25, 62, 68, 69, 72–74, 76–78, 80, 87, 104, 112–119, 121, 122, 165

RMSE Root-Mean-Square Error 67

UBKP Unconstrained Binary Knapsack Problem 43, 44, 46, 47, 49, 50

Notation

$\mathcal{X}$	Feasible solutions set
$\mathcal{X}_E$	Efficient set
$\mathcal{X}_S$	Supported solutions set
$\mathcal{X}_{ES}$	Extreme supported solutions set
$\mathcal{Y}$	Objective vectors set
$\mathcal{Y}_N$	Non-dominated set
$\mathcal{Y}_S$	Supported objective vectors set
$\mathcal{Y}_{ES}$	Extreme supported objective vectors set
$y^1 \geq y^2$	Point y^1 weakly dominates y^2
$y^1 > y^2$	Point y^1 dominates y^2
$y^1 \succ y^2$	Point y^1 strictly dominates y^2
$y^1 \geq_{\text{lex}} y^2$	Point y^1 is lexicographically greater than or equal to y^2
$y^1 >_{\text{lex}} y^2$	Point y^1 is lexicographically greater than y^2
$Y^1 \geq Y^2$	Set Y^1 weakly dominates Y^2
$Y^1 > Y^2$	Set Y^1 dominates Y^2
$Y^1 \succ Y^2$	Set Y^1 strictly dominates Y^2
I_H	Hypervolume indicator
I_ε	ε-indicator
$\mathbb{R}$	The set of real numbers
$\mathbb{Z}$	The set of integers

List of References

[1] Y. P. Aneja and K. P. K. Nair. “Bicriteria Transportation Problem”. In: *Management Science* 25.1 (1979), pp. 73–78. DOI: 10.1287/mnsc.25.1.73 (cit. on pp. 16, 17).

[2] A. Arbelaez, Y. Hamadi, and M. Sebag. “Online Heuristic Selection in Constraint Programming”. In: *International Symposium on Combinatorial Search*. 2009 (cit. on p. 29).

[3] C. Bazgan, H. Hugot, and D. Vanderpooten. “Solving Efficiently the 0-1 Multi-Objective Knapsack Problem”. In: *Computers & Operations Research* 36.1 (2009), pp. 260–279. DOI: 10.1016/j.cor.2007.09.009 (cit. on pp. 21, 23, 64, 68).

[4] *benchmark*. Version v1.6.0. Google (cit. on p. 42).

[5] F. Bökler. “Output-Sensitive Complexity of Multiobjective Combinatorial Optimization with an Application to the Multiobjective Shortest Path Problem”. PhD thesis. Technischen Universität Dortmund, 2018 (cit. on p. 17).

[6] A. Cerqueus, X. Gandibleux, A. Przybylski, and F. Saubion. “On Branching Heuristics for the Bi-Objective 0/1 Unidimensional Knapsack Problem”. In: *Journal of Heuristics* 23.5 (2017), pp. 285–319. DOI: 10.1007/s10732-017-9346-9 (cit. on p. 21).

[7] M. Chiarandini. “Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems”. PhD thesis. Darmstadt: Technische Universität, 2005 (cit. on p. 27).

[8] W. S. Cleveland, E. Grosse, and W. M. Shyu. “Local Regression Models”. In: *Statistical Models in S*. Routledge, 1992 (cit. on p. 70).

[9] G. B. Dantzig. “Discrete-Variable Extremum Problems”. In: *Operations Research* 5.2 (1957), pp. 266–288. DOI: 10.1287/opre.5.2.266 (cit. on pp. 21, 23).

[10] F. Daolio, A. Liefooghe, S. Verel, H. Aguirre, and K. Tanaka. “Problem Features versus Algorithm Performance on Rugged Multiobjective Combinatorial Fitness Landscapes”. In: *Evolutionary Computation* 25.4 (2017), pp. 555–585. DOI: 10.1162/evco_a_00193 (cit. on p. 28).

[11] T. Dean and M. Boddy. “An Analysis of Time-Dependent Planning”. In: *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*. AAAI’88. AAAI Press, 1988, pp. 49–54 (cit. on pp. 2, 25).

[12] C. Delort and O. Spanjaard. “Using Bound Sets in Multiobjective Optimization: Application to the Biobjective Binary Knapsack Problem”. In: *Experimental Algorithms*. SEA 2010. Vol. 6049. Lecture Notes in Computer Science. Springer, 2010, pp. 253–265. DOI: 10.1007/978-3-642-13193-6_22 (cit. on p. 23).

[13] D. M. Dias, A. D. Jesus, and L. Paquete. “A Software Library for Archiving Nondominated Points”. In: *Proceedings of the 2021 Genetic and Evolutionary Computation Conference Companion*. GECCO 2021. Association for Computing Machinery, 2021, pp. 53–54. DOI: 10.1145/3449726.3462737 (cit. on p. 7).

[14] D. M. Dias, A. D. Jesus, and L. Paquete. *nondLib*. Version v0.2.0. 2021. DOI: 10.5281/zenodo.4733027 (cit. on p. 7).

[15] M. Ehrgott. “Hard to Say It’s Easy — Four Reasons Why Combinatorial Multiobjective Programmes Are Hard”. In: *Research and Practice in Multiple Criteria Decision Making*. MCDM. Vol. 487. Lecture Notes in Economics and Mathematical Systems. Springer, 2000, pp. 69–80. DOI: 10.1007/978-3-642-57311-8_5 (cit. on p. 1).

[16] M. Ehrgott. *Multicriteria Optimization*. 2nd ed. Springer, 2005. DOI: 10.1007/3-540-27659-9 (cit. on pp. 10, 11, 16, 17, 49).

[17] M. Ehrgott and X. Gandibleux. “Bounds and Bound Sets for Biobjective Combinatorial Optimization Problems”. In: *Multiple Criteria Decision Making in the New Millennium*. MCDM. Vol. 507. Lecture Notes in Economics and Mathematical Systems. Springer, 2001, pp. 241–253. DOI: 10.1007/978-3-642-56680-6_22 (cit. on p. 18).

[18] M. T. M. Emmerich and A. H. Deutz. “Test Problems Based on Lamé Superspheres”. In: *Evolutionary Multi-Criterion Optimization*. EMO 2007. Vol. 4403. Lecture Notes in Computer Science. Springer, 2007, pp. 922–936. DOI: 10.1007/978-3-540-70928-2_68 (cit. on p. 33).

[19] J. R. Figueira, L. Paquete, M. Simões, and D. Vanderpooten. “Algorithmic Improvements on Dynamic Programming for the Bi-Objective {0,1} Knapsack Problem”. In: *Computational Optimization and Applications* 56.1 (2013), pp. 97–111. DOI: 10.1007/s10589-013-9551-x (cit. on pp. 16, 23, 24).

[20] G. Fitzmaurice, M. Davidian, G. Verbeke, and G. Molenberghs, eds. *Longitudinal Data Analysis.* CRC Press, 2008 (cit. on p. 60).

[21] M. Gagliolo, C. Legrand, and M. Birattari. “Mixed-Effects Modeling of Optimisation Algorithm Performance”. In: *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics.* SLS 2009. Vol. 5752. Lecture Notes in Computer Science. Springer, 2009, pp. 150–154. DOI: 10.1007/978-3-642-03751-1_17 (cit. on pp. 60, 61).

[22] M. Gagliolo and J. Schmidhuber. “Learning Dynamic Algorithm Portfolios”. In: *Annals of Mathematics and Artificial Intelligence* 47.3 (2006), pp. 295–328. DOI: 10.1007/s10472-006-9036-z (cit. on p. 29).

[23] M. Gagliolo, V. Zhumatiy, and J. Schmidhuber. “Adaptive Online Time Allocation to Search Algorithms”. In: *Machine Learning: ECML 2004.* ECML 2004. Vol. 3201. Lecture Notes in Computer Science. Springer, 2004, pp. 134–143. DOI: 10.1007/978-3-540-30115-8_15 (cit. on p. 29).

[24] C. E. Galarza, L. M. Castro, F. Louzada, and V. H. Lachos. “Quantile Regression for Nonlinear Mixed Effects Models: A Likelihood Based Perspective”. In: *Statistical Papers* 61.3 (2020), pp. 1281–1307. DOI: 10.1007/s00362-018-0988-y (cit. on p. 61).

[25] V. Grunert da Fonseca, C. M. Fonseca, and A. O. Hall. “Inferential Performance Assessment of Stochastic Optimisers and the Attainment Function”. In: *Evolutionary Multi-Criterion Optimization.* EMO 2001. Vol. 1993. Lecture Notes in Computer Science. Springer, 2001, pp. 213–225. DOI: 10.1007/3-540-44719-9_15 (cit. on p. 27).

[26] A. P. Guerreiro and C. M. Fonseca. “Computing and Updating Hypervolume Contributions in Up to Four Dimensions”. In: *IEEE Transactions on Evolutionary Computation* 22.3 (2018), pp. 449–463. DOI: 10.1109/TEVC.2017.2729550 (cit. on p. 130).

[27] A. P. Guerreiro, C. M. Fonseca, and L. Paquete. “The Hypervolume Indicator: Computational Problems and Algorithms”. In: *ACM Computing Surveys* 54.6 (2021), 119:1–119:42. DOI: 10.1145/3453474 (cit. on p. 129).

[28] E. A. Hansen and S. Zilberstein. “Monitoring and Control of Anytime Algorithms: A Dynamic Programming Approach”. In: *Artificial Intelligence* 126.1 (2001), pp. 139–157. DOI: 10.1016/S0004-3702(00)00068-0 (cit. on p. 29).

[29] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., 2005 (cit. on pp. 27, 28).

[30] A. D. Jesus. *anytime*. Version v0.0.2. Zenodo, 2022. DOI: 10.5281/zenodo.6856120 (cit. on p. 8).

[31] A. D. Jesus. *apm*. Version v0.1.1. Zenodo, 2022. DOI: 10.5281/zenodo.6857541 (cit. on pp. 8, 42).

[32] A. D. Jesus. *Data, Scripts, and Results for: Algorithm Selection for Multi-Objective Optimization*. Zenodo, 2022. DOI: 10.5281/zenodo.6858045 (cit. on pp. 8, 43, 68, 69, 112, 113, 132, 147).

[33] A. D. Jesus. *mobkp*. Version v0.1.1. Zenodo, 2022. DOI: 10.5281/zenodo.6857821 (cit. on pp. 7, 16, 21, 68, 112, 132, 147).

[34] A. D. Jesus. *moco_abm*. Version v0.2.0. Zenodo, 2019. DOI: 10.5281/zenodo.3548869 (cit. on p. 7).

[35] A. D. Jesus. *mooutils*. Version v0.1.0. Zenodo, 2022. DOI: 10.5281/zenodo.6855879 (cit. on p. 7).

[36] A. D. Jesus, A. Liefooghe, B. Derbel, and L. Paquete. “Algorithm Selection of Anytime Algorithms”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO 2020. Association for Computing Machinery, 2020, pp. 850–858. DOI: 10.1145/3377930.3390185 (cit. on pp. 6, 64, 109).

[37] A. D. Jesus, L. Paquete, B. Derbel, and A. Liefooghe. “On the Design and Anytime Performance of Indicator-based Branch and Bound for Multi-objective Combinatorial Optimization”. In: *Proceedings of the 2021 Genetic and Evolutionary Computation Conference*. GECCO 2021. Association for Computing Machinery, 2021, pp. 234–242. DOI: 10.1145/3449639.3459360 (cit. on pp. 6, 127).

[38] A. D. Jesus, L. Paquete, and A. Liefooghe. “A Model of Anytime Algorithm Performance for Bi-Objective Optimization”. In: *Journal of Global Optimization* 79 (2020), pp. 329–350. DOI: 10.1007/s10898-020-00909-9 (cit. on pp. 6, 31).

[39] A. D. Jesus, L. Paquete, and A. Liefooghe. “A Model of Anytime Algorithm Performance for Biobjective Optimization Problems”. In: *Proceedings LeGO - 14th International Global Optimization Workshop*. LeGO 2018. Vol. 2070. AIP Conference Proceedings. AIP, 2019, p. 020049. DOI: 10.1063/1.5090016 (cit. on pp. 6, 31).

[40] A. D. Jesus, L. Paquete, A. Liefooghe, and B. Derbel. “Techniques to Analyze the Anytime Behavior of Algorithms for Multi-Objective Optimization”. 31st European Conference on Operational Research (EURO 2021). 2021 (cit. on p. 7).

[41] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. “Automated Algorithm Selection: Survey and Perspectives”. In: *Evolutionary Computation* 27.1 (2019), pp. 3–45. DOI: 10.1162/evco_a_00242 (cit. on pp. 2, 28).

[42] K. Klamroth and M. M. Wiecek. “Dynamic Programming Approaches to the Multiple Criteria Knapsack Problem”. In: *Naval Research Logistics (NRL)* 47.1 (2000), pp. 57–76. DOI: 10.1002/(SICI)1520-6750(200002)47:1<57::AID-NAV4>3.0.CO;2-4 (cit. on p. 23).

[43] J. Knowles and D. Corne. “On Metrics for Comparing Nondominated Sets”. In: *Proceedings of the 2002 Congress on Evolutionary Computation*. CEC 2002. IEEE, 2002, pp. 711–716. DOI: 10.1109/CEC.2002.1007013 (cit. on pp. 11, 12).

[44] L. Kotthoff. “Algorithm Selection for Combinatorial Search Problems: A Survey”. In: *Data Mining and Constraint Programming*. Vol. 10101. Lecture Notes in Computer Science. Springer, 2016, pp. 149–190. DOI: 10.1007/978-3-319-50137-6_7 (cit. on pp. 2, 28, 29).

[45] M. Laumanns, L. Thiele, and E. Zitzler. “An Efficient, Adaptive Parameter Variation Scheme for Metaheuristics Based on the Epsilon-Constraint Method”. In: *European Journal of Operational Research* 169.3 (2006), pp. 932–942. DOI: 10.1016/j.ejor.2004.08.029 (cit. on p. 18).

[46] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. “A Portfolio Approach to Algorithm Selection”. In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI ’03. Morgan Kaufmann Publishers Inc., 2003, pp. 1542–1543 (cit. on p. 28).

[47] A. Liefooghe, F. Daolio, S. Verel, B. Derbel, H. Aguirre, and K. Tanaka. “Landscape-Aware Performance Prediction for Evolutionary Multiobjective Optimization”. In: *IEEE Transactions on Evolutionary Computation* 24.6 (2020), pp. 1063–1077. DOI: 10.1109/TEVC.2019.2940828 (cit. on p. 28).

[48] A. Liefooghe, L. Paquete, M. Simões, and J. R. Figueira. "Connectedness and Local Search for Bicriteria Knapsack Problems". In: *Evolutionary Computation in Combinatorial Optimization*. EvoCOP 2011. Vol. 6622. Lecture Notes in Computer Science. Springer, 2011, pp. 48–59. DOI: 10.1007/978-3-642-20364-0_5 (cit. on pp. 24, 25).

[49] A. Liefooghe, S. Verel, B. Lacroix, A.-C. Zăvoianu, and J. McCall. "Landscape Features and Automated Algorithm Selection for Multi-Objective Interpolated Continuous Optimisation Problems". In: *Proceedings of the 2021 Genetic and Evolutionary Computation Conference*. GECCO 2021. Association for Computing Machinery, 2021, pp. 421–429. DOI: 10.1145/3449639.3459353 (cit. on p. 28).

[50] M. López-Ibáñez, L. Paquete, and T. Stützle. "Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization". In: *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, 2010, pp. 209–222. DOI: 10.1007/978-3-642-02538-9_9 (cit. on p. 28).

[51] M. López-Ibáñez and T. Stützle. "Automatically Improving the Anytime Behaviour of Optimisation Algorithms". In: *European Journal of Operational Research* 235.3 (2014), pp. 569–582. DOI: 10.1016/j.ejor.2013.10.043 (cit. on pp. 64, 109).

[52] U.-E. Lukata and J. Teghem. "Solving Multi-Objective Knapsack Problem by a Branch-and-Bound Procedure". In: *Multicriteria Analysis*. MCDM. Springer, 1997, pp. 269–278. DOI: 10.1007/978-3-642-60667-0_26 (cit. on pp. 19, 20, 22).

[53] A. Makhorin. *GLPK*. Version v5.0. 2020 (cit. on p. 49).

[54] S. Martello and P. Toth. "An Upper Bound for the Zero-One Knapsack Problem and a Branch and Bound Algorithm". In: *European Journal of Operational Research* 1.3 (1977), pp. 169–175. DOI: 10.1016/0377-2217(77)90024-8 (cit. on pp. 22, 24).

[55] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph. "Exploratory Landscape Analysis". In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO 2011. Association for Computing Machinery, 2011, pp. 829–836. DOI: 10.1145/2001576.2001690 (cit. on p. 28).

[56] O. Mersmann, B. Bischl, H. Trautmann, M. Wagner, J. Bossek, and F. Neumann. "A Novel Feature-Based Approach to Characterize Algorithm Performance for the Traveling Salesperson Problem". In: *Annals of Mathematics and Artificial Intelligence* 69.2 (2013), pp. 151–182. DOI: 10.1007/s10472-013-9341-2 (cit. on p. 28).

[57] A. M. Molinaro, R. Simon, and R. M. Pfeiffer. "Prediction Error Estimation: A Comparison of Resampling Methods". In: *Bioinformatics* 21.15 (2005), pp. 3301–3307. DOI: 10.1093/bioinformatics/bti499 (cit. on p. 67).

[58] G. L. Nemhauser and Z. Ullmann. "Discrete Dynamic Programming and Capital Allocation". In: *Management Science* 15.9 (1969), pp. 494–505. DOI: 10.1287/mnsc.15.9.494 (cit. on pp. 23, 44).

[59] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. "Understanding Random SAT: Beyond the Clauses-to-Variables Ratio". In: *Principles and Practice of Constraint Programming - CP 2004*. CP 2004. Vol. 3258. Lecture Notes in Computer Science. Springer, 2004, pp. 438–452. DOI: 10.1007/978-3-540-30201-8_33 (cit. on p. 28).

[60] L. Paquete, M. Chiarandini, and T. Stützle. "Pareto Local Optimum Sets in the Biobjective Traveling Salesman Problem: An Experimental Study". In: *Metaheuristics for Multiobjective Optimisation*. Vol. 535. Lecture Notes in Economics and Mathematical Systems. Springer, 2004, pp. 177–199. DOI: 10.1007/978-3-642-17144-4_7 (cit. on p. 24).

[61] L. Paquete, B. Schulze, M. Stiglmayr, and A. C. Lourenço. "Computing Representations Using Hypervolume Scalarizations". In: *Computers & Operations Research* 137 (2022), p. 105349. DOI: 10.1016/j.cor.2021.105349 (cit. on pp. 55, 163).

[62] M. Pilu and R. B. Fisher. "Equal-Distance Sampling of Superellipse Models". In: *Procedings of the 1995 British Machine Vision Conference*. BMVC 1995. BMVA Press, 1995, pp. 257–266. DOI: 10.5244/C.9.26 (cit. on p. 34).

[63] J. Pinheiro and D. Bates. *Mixed-Effects Models in S and S-PLUS*. Statistics and Computing. Springer, 2000. DOI: 10.1007/b98882 (cit. on p. 60).

[64] S. Polyakovskiy, M. R. Bonyadi, M. Wagner, Z. Michalewicz, and F. Neumann. "A Comprehensive Benchmark Set and Heuristics for the Traveling Thief Problem". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. GECCO 2014. Association for Computing Machinery, 2014, pp. 477–484. DOI: 10.1145/2576768.2598249 (cit. on p. 28).

[65] A. Przybylski and X. Gandibleux. "Multi-Objective Branch and Bound". In: *European Journal of Operational Research* 260.3 (2017), pp. 856–872. DOI: 10.1016/j.ejor.2017.01.032 (cit. on pp. 18, 19).

[66] A. Przybylski, X. Gandibleux, and M. Ehrgott. “A Recursive Algorithm for Finding All Nondominated Extreme Points in the Outcome Set of a Multiobjective Integer Programme”. In: *INFORMS Journal on Computing* 22.3 (2010), pp. 371–386. DOI: 10.1287/ijoc.1090.0342 (cit. on pp. 17, 24).

[67] J. R. Rice. “The Algorithm Selection Problem”. In: *Advances in Computers* 15 (1976), pp. 65–118. DOI: 10.1016/S0065-2458(08)60520-3 (cit. on pp. 2, 28).

[68] S. J. Russell and S. Zilberstein. “Composing Real-time Systems”. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence*. IJCAI ’91. Morgan Kaufmann Publishers Inc., 1991, pp. 212–217 (cit. on p. 29).

[69] K. A. Smith-Miles. “Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection”. In: *ACM Computing Surveys (CSUR)* 41.1 (2008), p. 6. DOI: 10.1145/1456650.1456656 (cit. on p. 2).

[70] V. Srinivasan and G. L. Thompson. “Algorithms for Minimizing Total Cost, Bottleneck Time and Bottleneck Shipment in Transportation Problems”. In: *Naval Research Logistics Quarterly* 23.4 (1976), pp. 567–595. DOI: 10.1002/nav.3800230402 (cit. on p. 17).

[71] J. Svegliato, K. H. Wray, and S. Zilberstein. “Meta-Level Control of Anytime Algorithms with Online Performance Prediction”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. IJCAI ’18. 2018. DOI: 10.24963/ijcai.2018/208 (cit. on p. 29).

[72] S. Verel, A. Liefooghe, L. Jourdan, and C. Dhaenens. “Analyzing the Effect of Objective Correlation on the Efficient Set of MNK-Landscapes”. In: *Learning and Intelligent Optimization*. LION 2011. Vol. 6683. Lecture Notes in Computer Science. Springer, 2011, pp. 116–130. DOI: 10.1007/978-3-642-25566-3_9 (cit. on p. 44).

[73] S. Verel, A. Liefooghe, L. Jourdan, and C. Dhaenens. “On the Structure of Multiobjective Combinatorial Search Space: MNK-landscapes with Correlated Objectives”. In: *European Journal of Operational Research* 227.2 (2013), pp. 331–342. DOI: 10.1016/j.ejor.2012.12.019 (cit. on p. 63).

[74] M. G. Vilas Boas, H. G. Santos, L. H. d. C. Merschmann, and G. V. Berghe. “Optimal Decision Trees for the Algorithm Selection Problem: Integer Programming Based Approaches”. In: *International Transactions in Operational Research* (2019). DOI: 10.1111/itor.12724 (cit. on p. 28).

[75] M. Visée, J. Teghem, M. Pirlot, and E. Ulungu. "Two-Phases Method and Branch and Bound Procedures to Solve the Bi–Objective Knapsack Problem". In: *Journal of Global Optimization* 12.2 (1998), pp. 139–155. DOI: 10.1023/A:1008258310679 (cit. on pp. 19, 20).

[76] L. While, L. Bradstreet, and L. Barone. "A Fast Way of Calculating Exact Hypervolumes". In: *IEEE Transactions on Evolutionary Computation* 16.1 (2012), pp. 86–95. DOI: 10.1109/TEVC.2010.2077298 (cit. on p. 130).

[77] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. "SATzilla: Portfolio-based Algorithm Selection for SAT". In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 565–606. DOI: 10.1613/jair.2490 (cit. on p. 28).

[78] S. Zilberstein. "Using Anytime Algorithms in Intelligent Systems". In: *AI Magazine* 17.3 (1996), pp. 73–83. DOI: 10.1609/aimag.v17i3.1232 (cit. on pp. 2, 25, 26, 28, 29).

[79] E. Zitzler. "Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications". PhD thesis. Swiss Federal Institute of Technology Zurich, 1999 (cit. on p. 13).

[80] E. Zitzler and L. Thiele. "Multiobjective Optimization Using Evolutionary Algorithms — A Comparative Case Study". In: *Parallel Problem Solving from Nature — PPSN*. PPSN 1998. Vol. 1498. Lecture Notes in Computer Science. Springer, 1998, pp. 292–301. DOI: 10.1007/BFb0056872 (cit. on pp. 3, 12).

[81] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. "Performance Assessment of Multiobjective Optimizers: An Analysis and Review". In: *IEEE Transactions on Evolutionary Computation* 7.2 (2003), pp. 117–132. DOI: 10.1109/TEVC.2003.810758 (cit. on pp. 11, 12, 14).

Appendix A

Empirical Model Results

A.1 PLS — 2 objectives

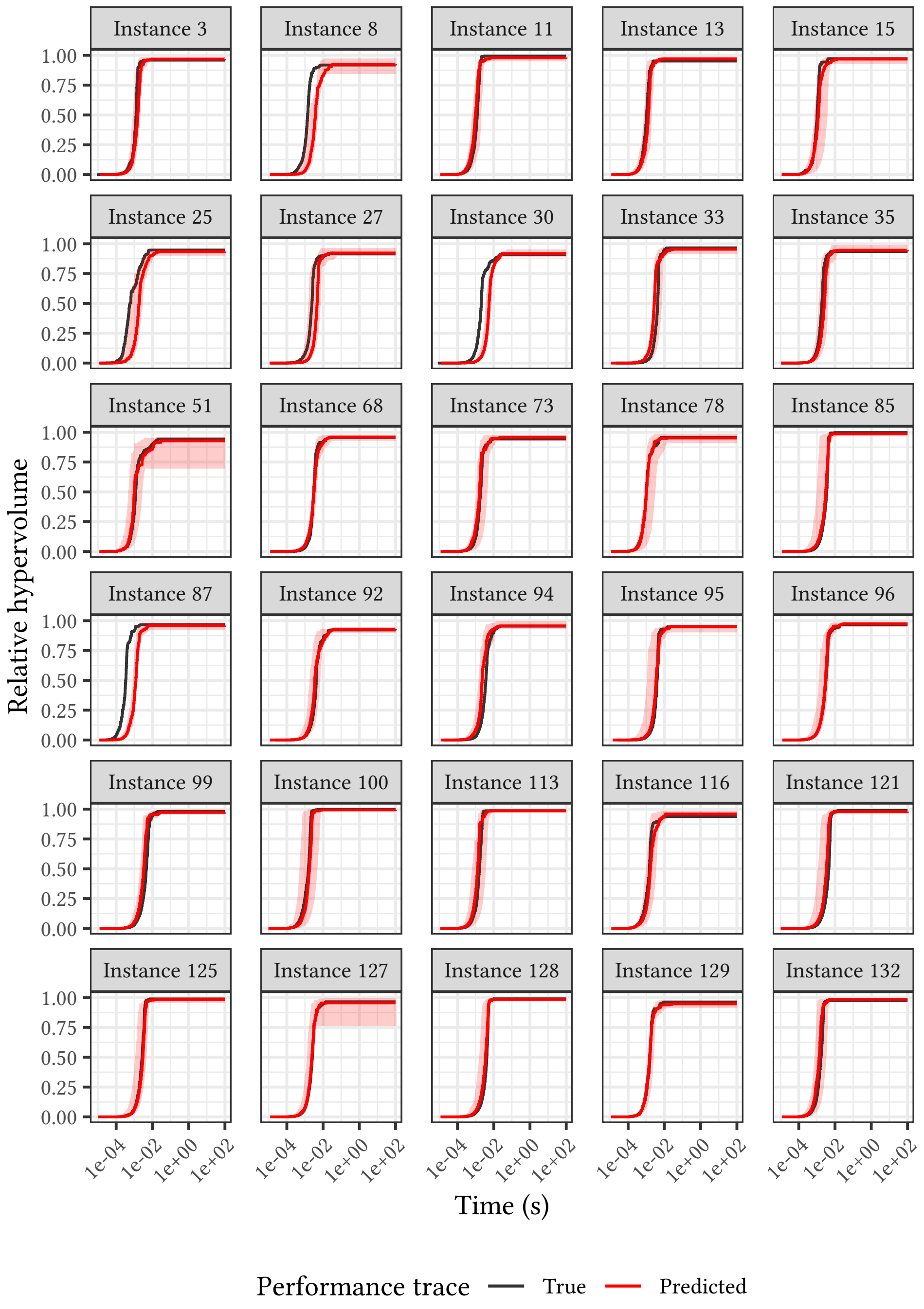


Figure A.1: Results of anytime performance prediction on the testing data set.

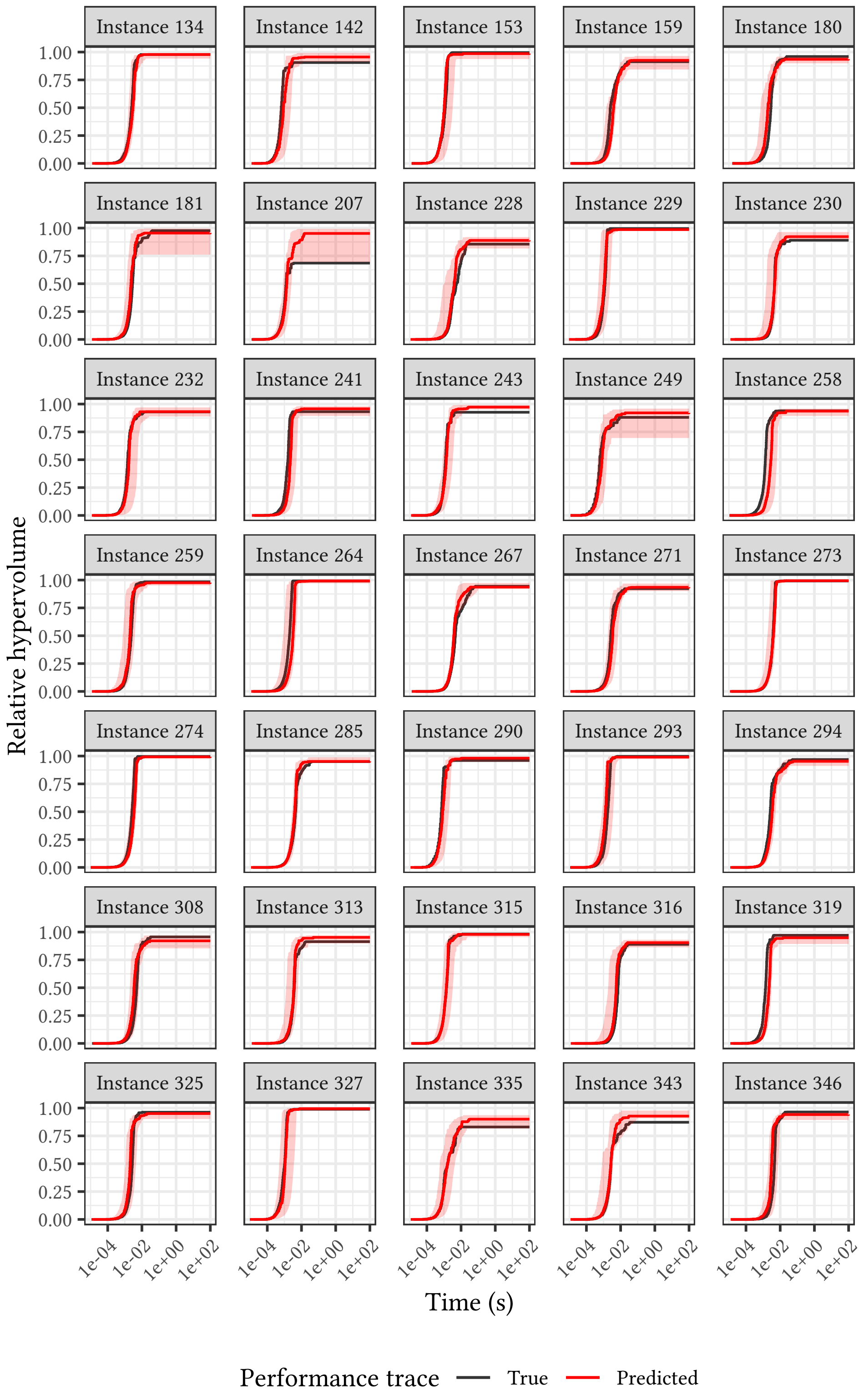


Figure A.2: Results of anytime performance prediction on the testing data set.

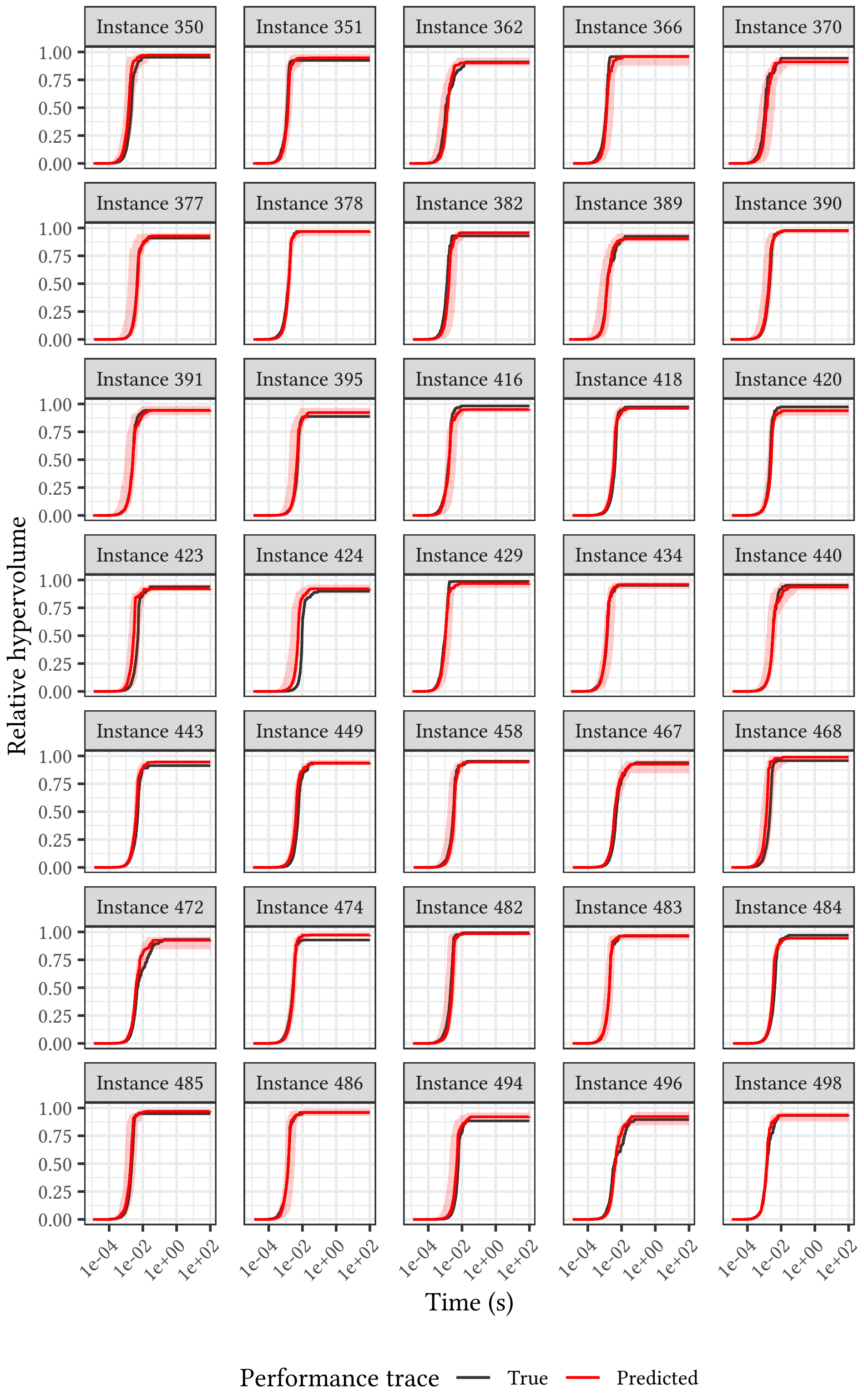


Figure A.3: Results of anytime performance prediction on the testing data set.

A.2 PLS — 3 objectives

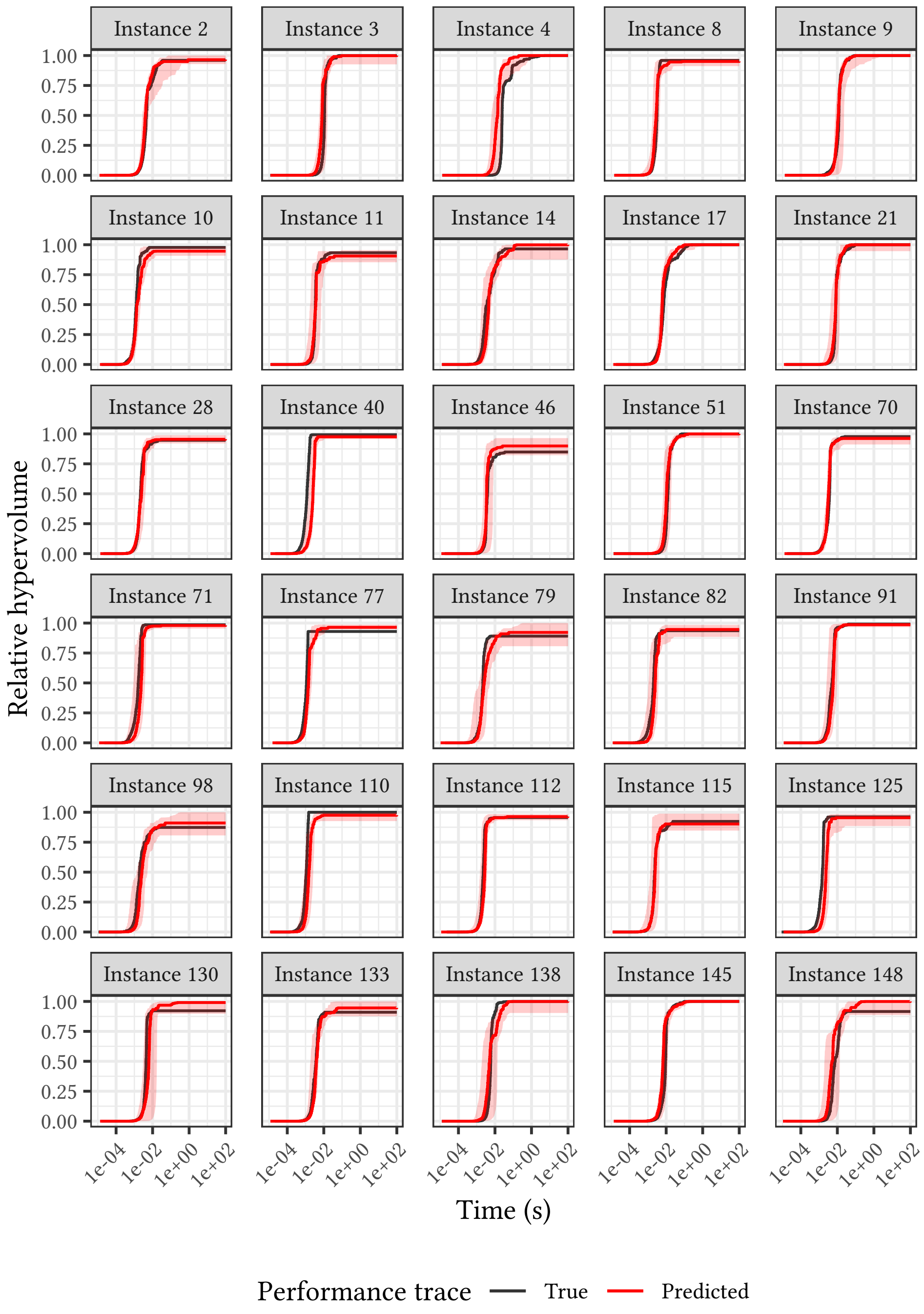


Figure A.4: Results of anytime performance prediction on the testing data set.

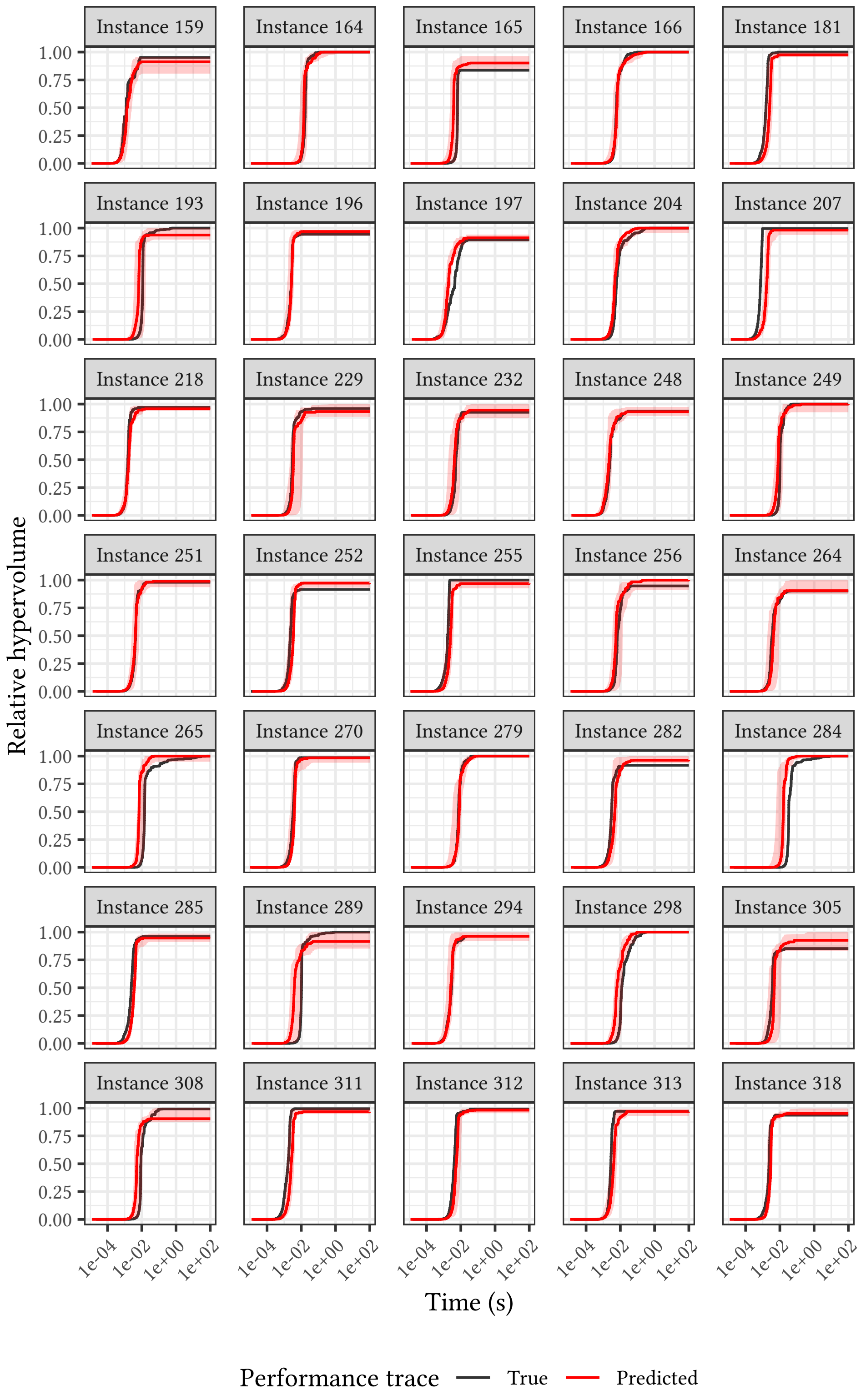


Figure A.5: Results of anytime performance prediction on the testing data set.

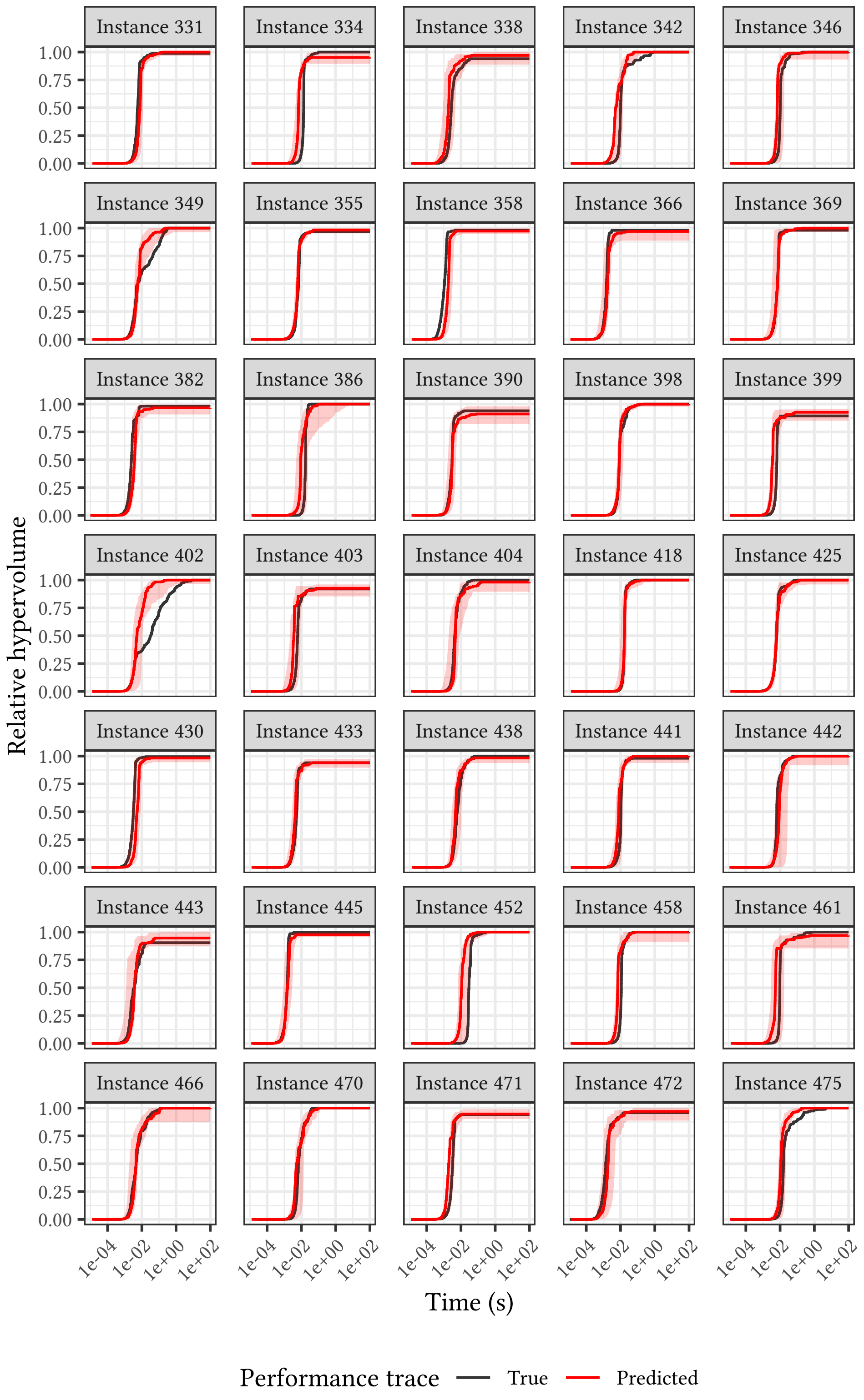


Figure A.6: Results of anytime performance prediction on the testing data set.

A.3 PLS — 5 objectives

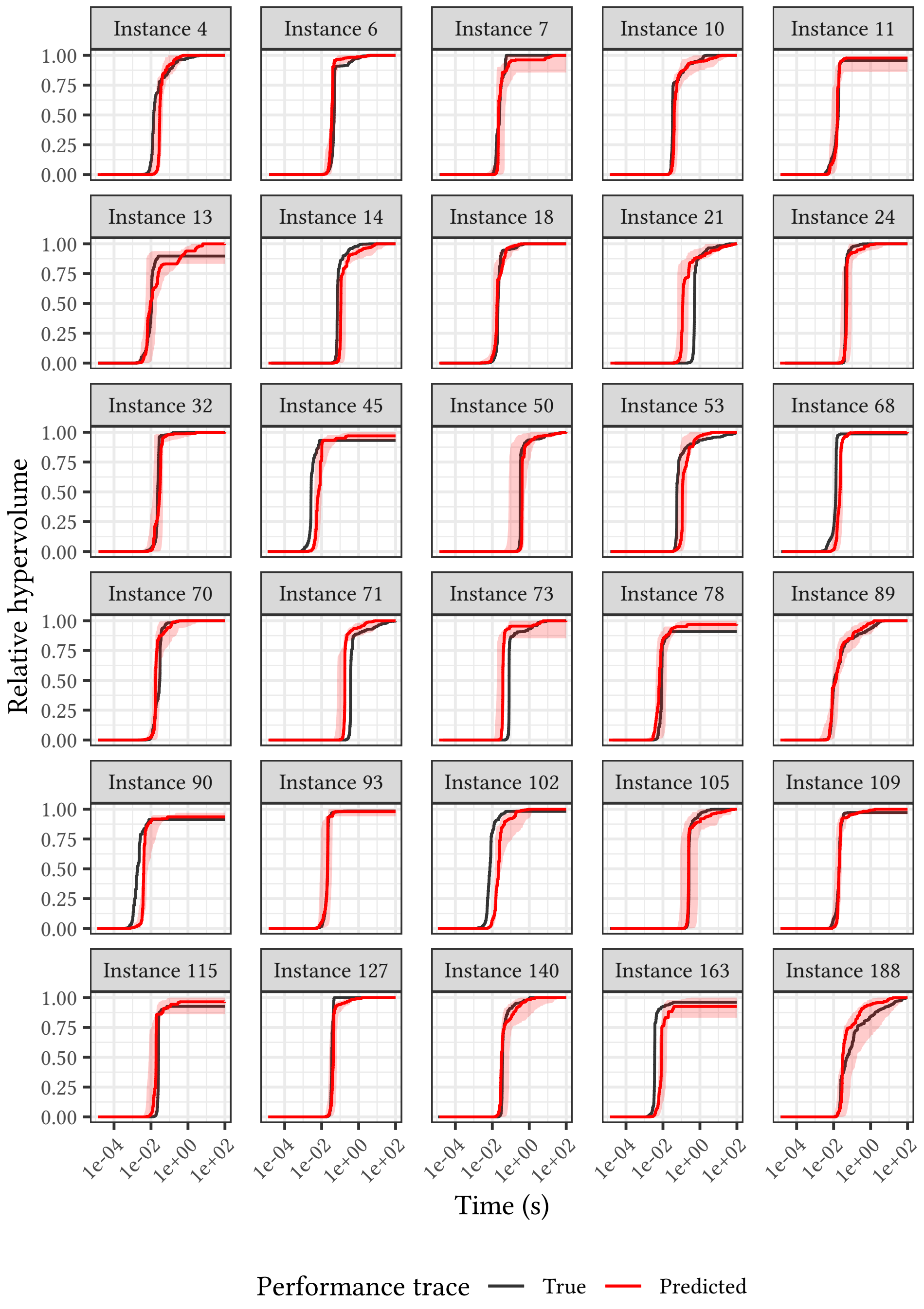


Figure A.7: Results of anytime performance prediction on the testing data set.

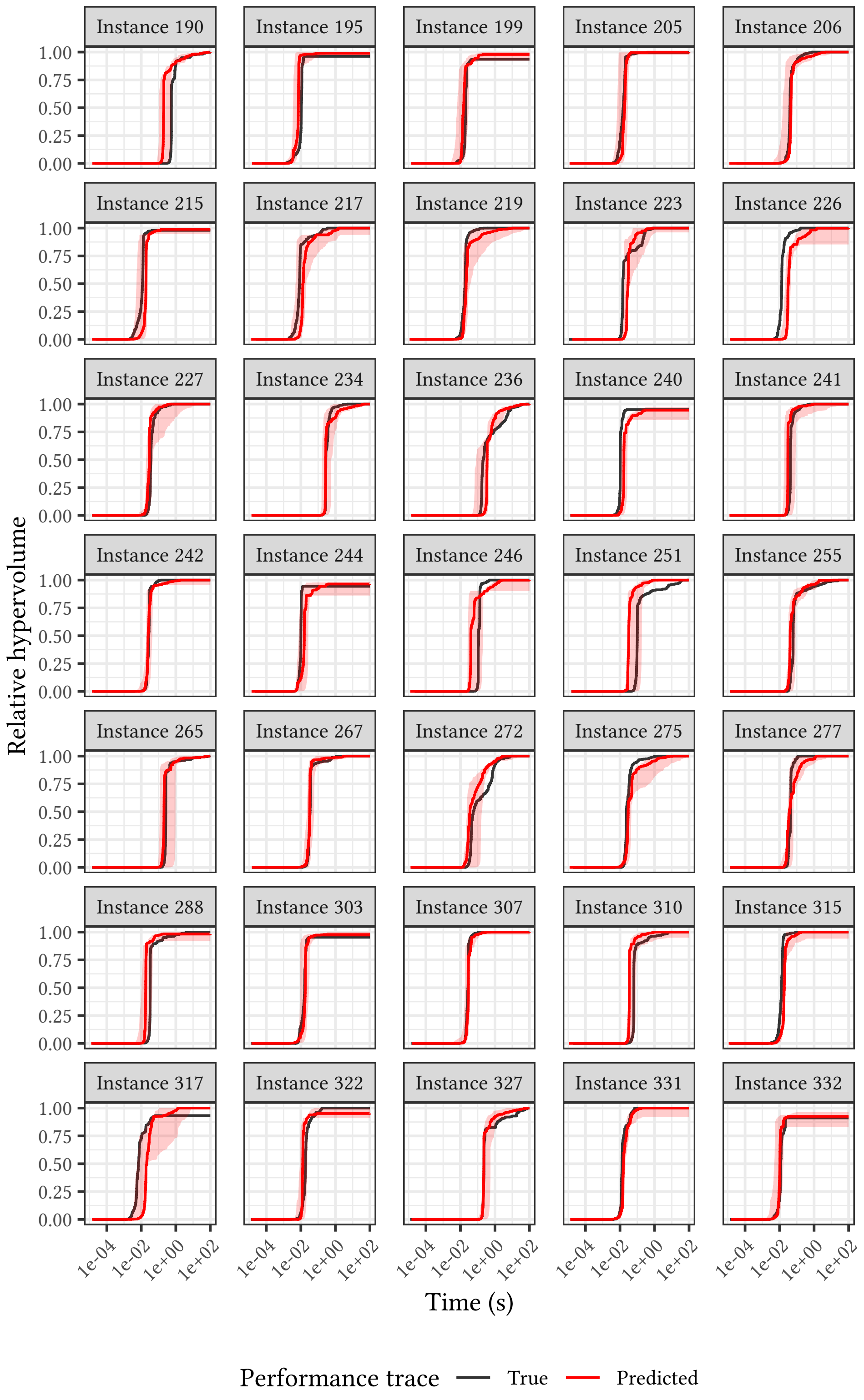


Figure A.8: Results of anytime performance prediction on the testing data set.

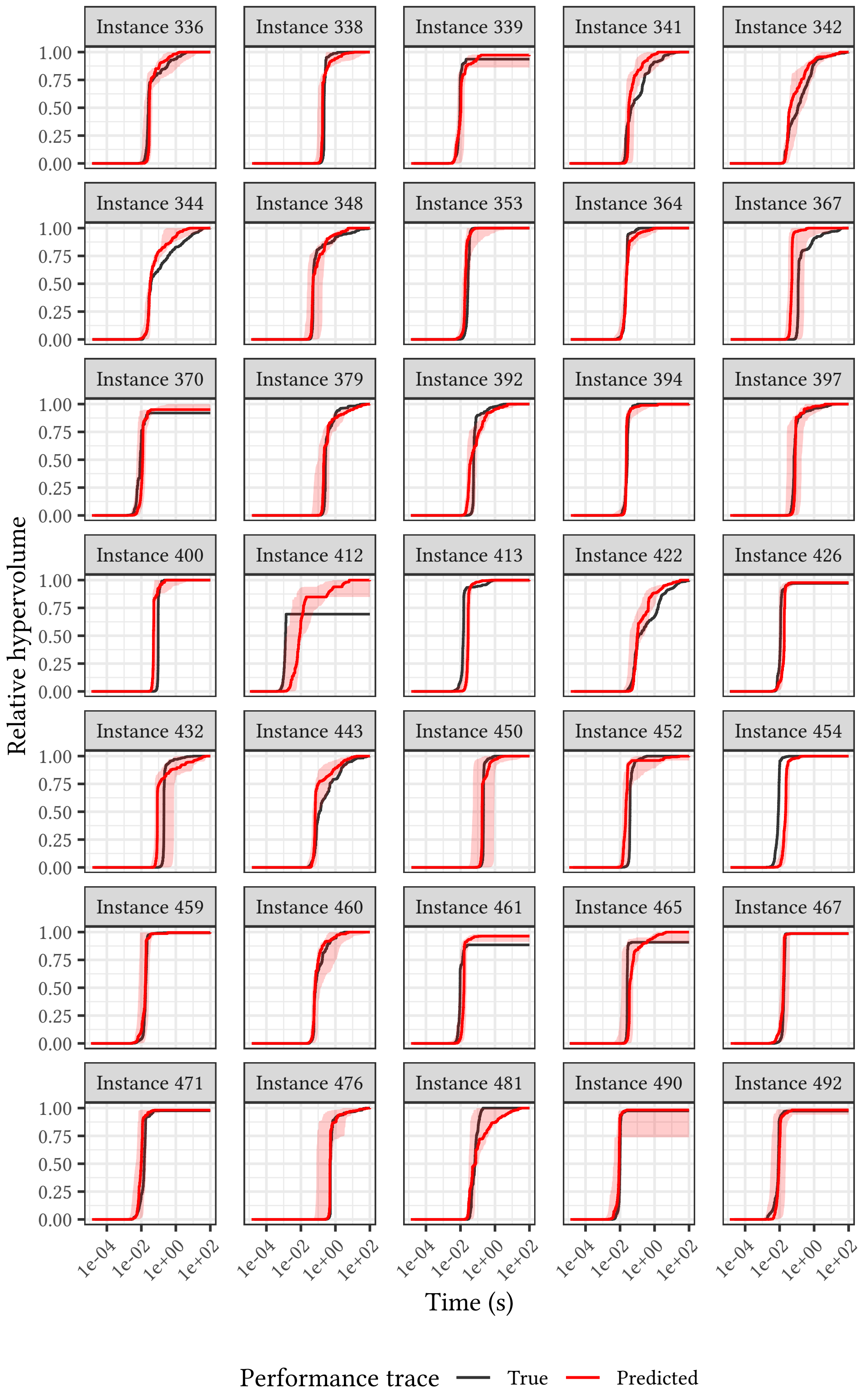


Figure A.9: Results of anytime performance prediction on the testing data set.

A.4 BHV-DP — 2 objectives

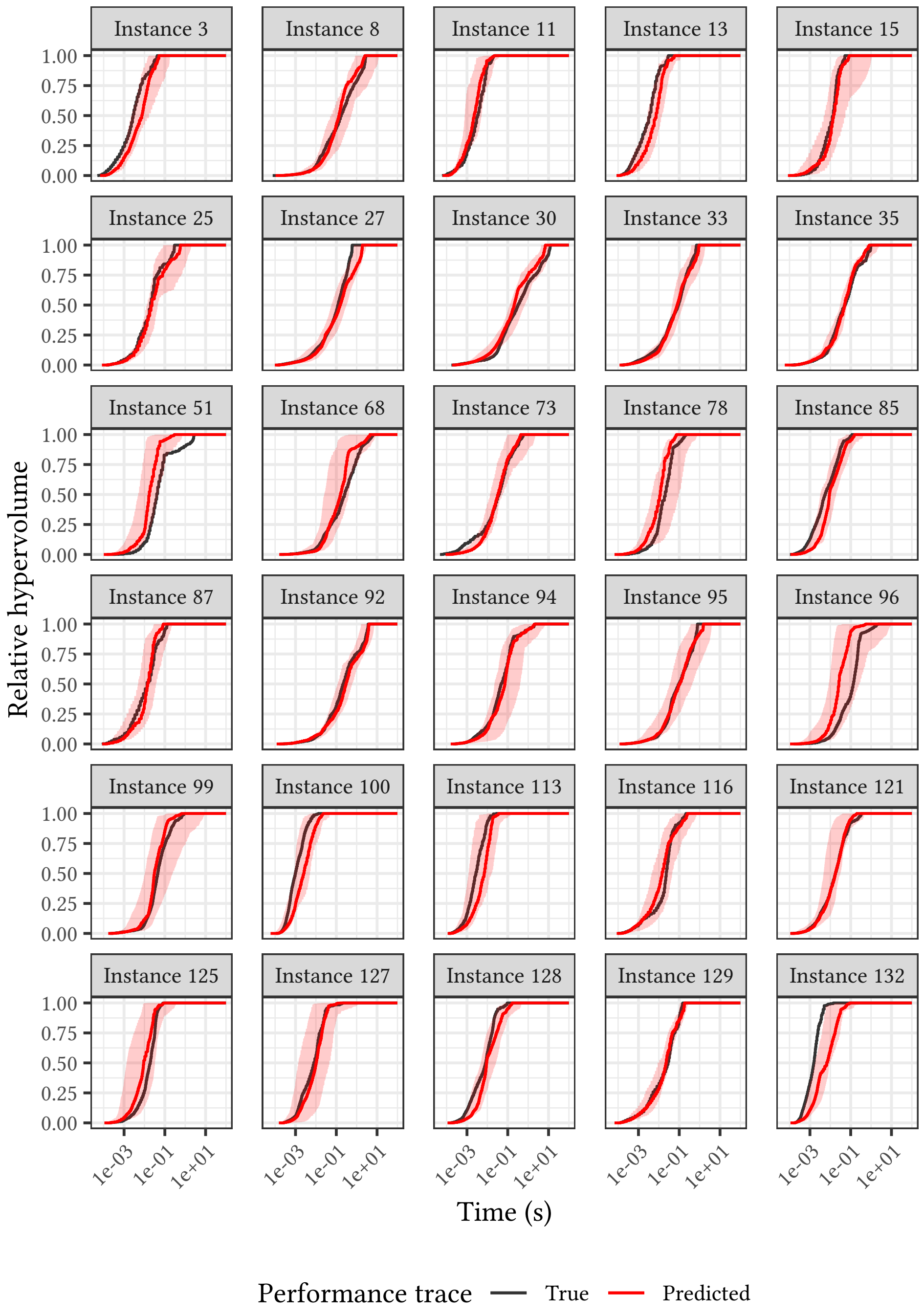


Figure A.10: Results of anytime performance prediction on the testing data set.

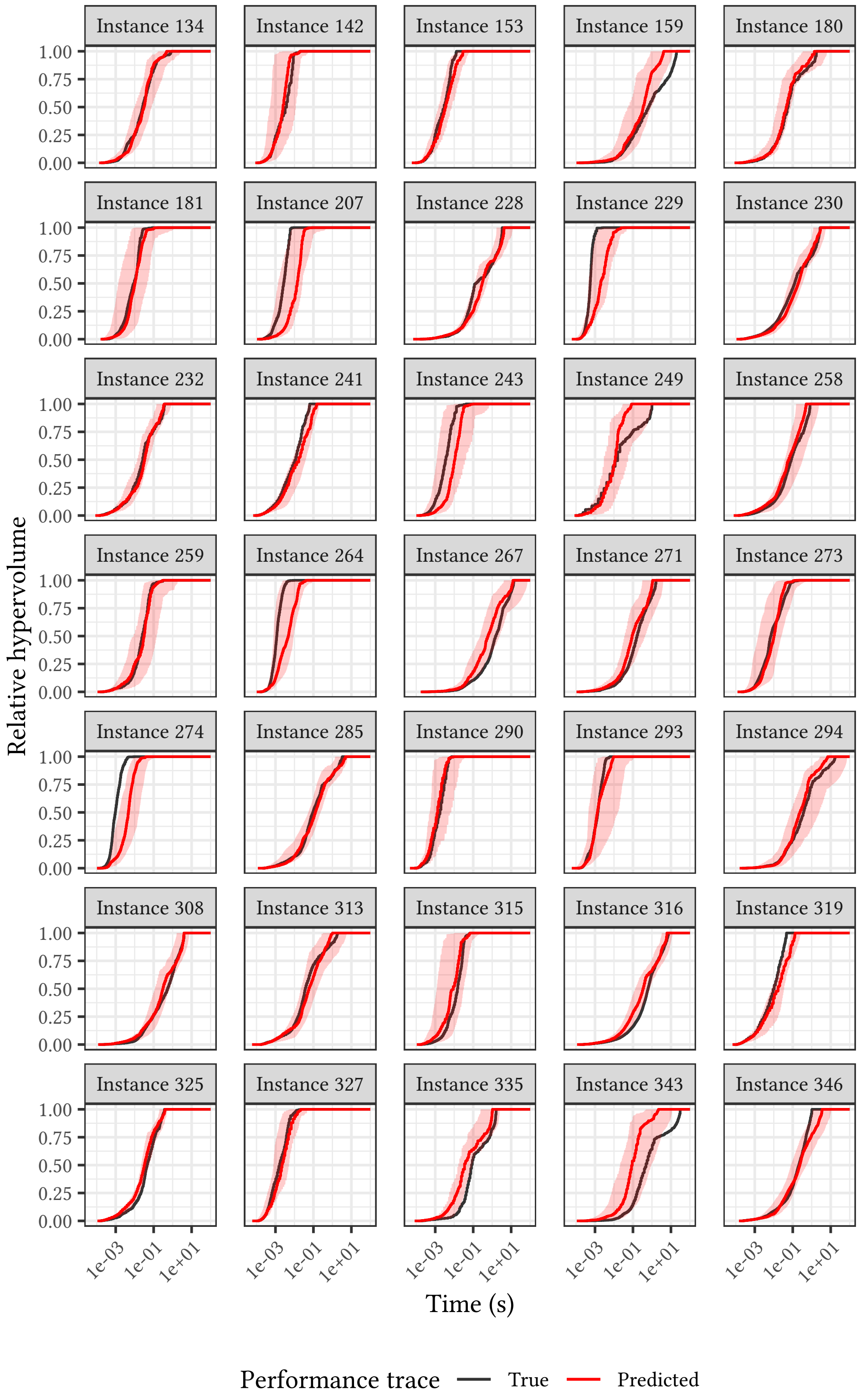


Figure A.11: Results of anytime performance prediction on the testing data set.

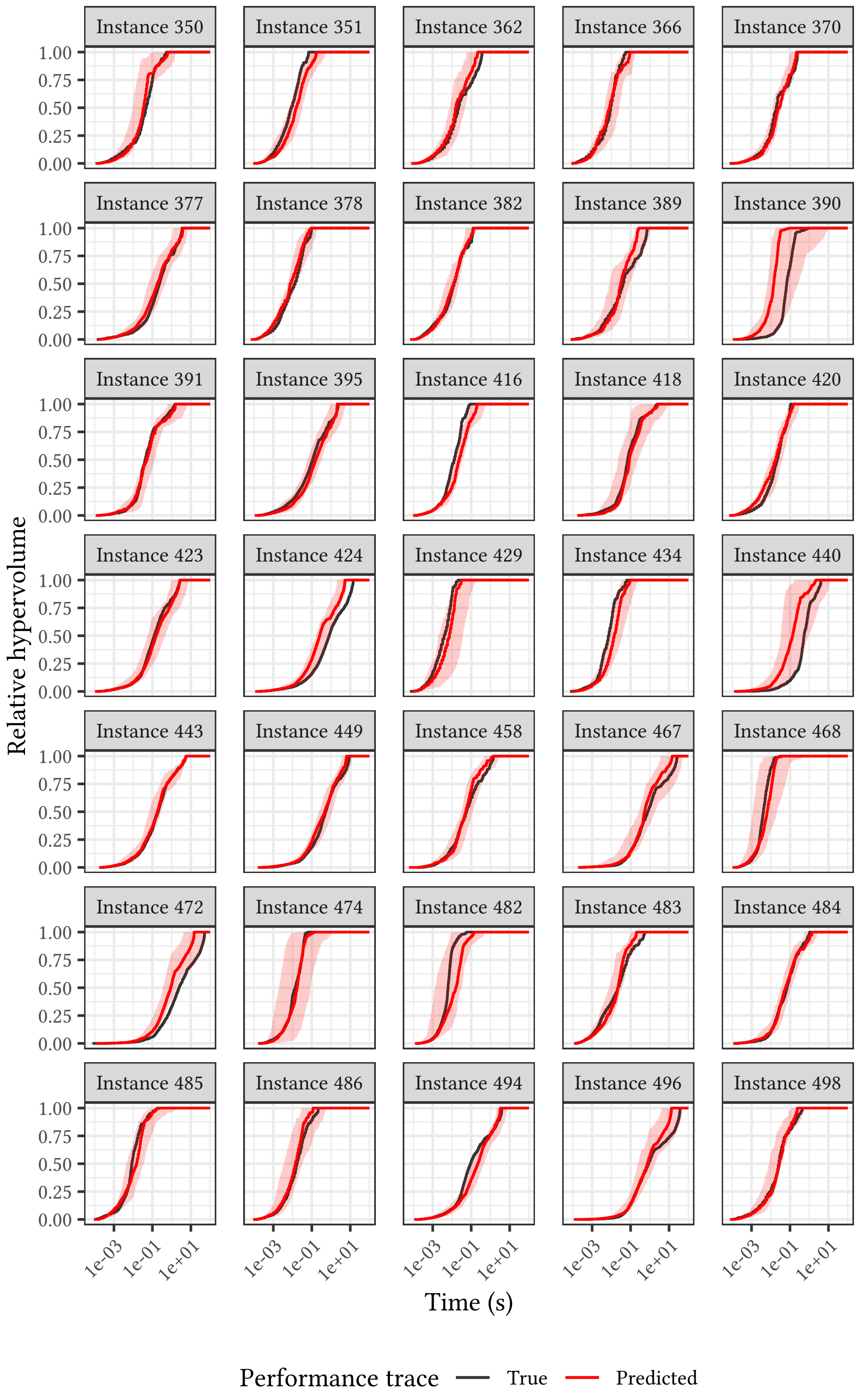


Figure A.12: Results of anytime performance prediction on the testing data set.

A.5 BHV-DP — 3 objectives

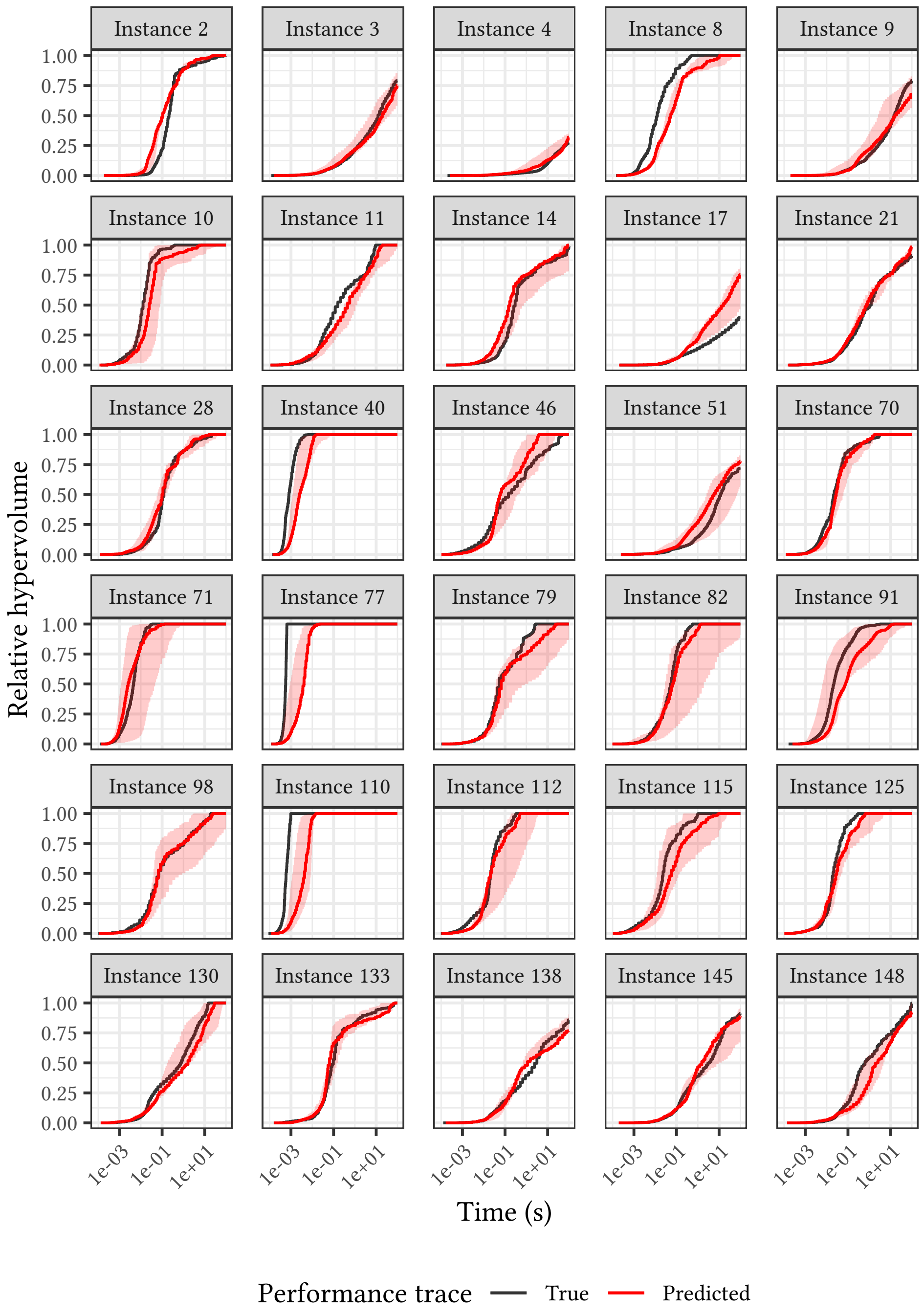


Figure A.13: Results of anytime performance prediction on the testing data set.

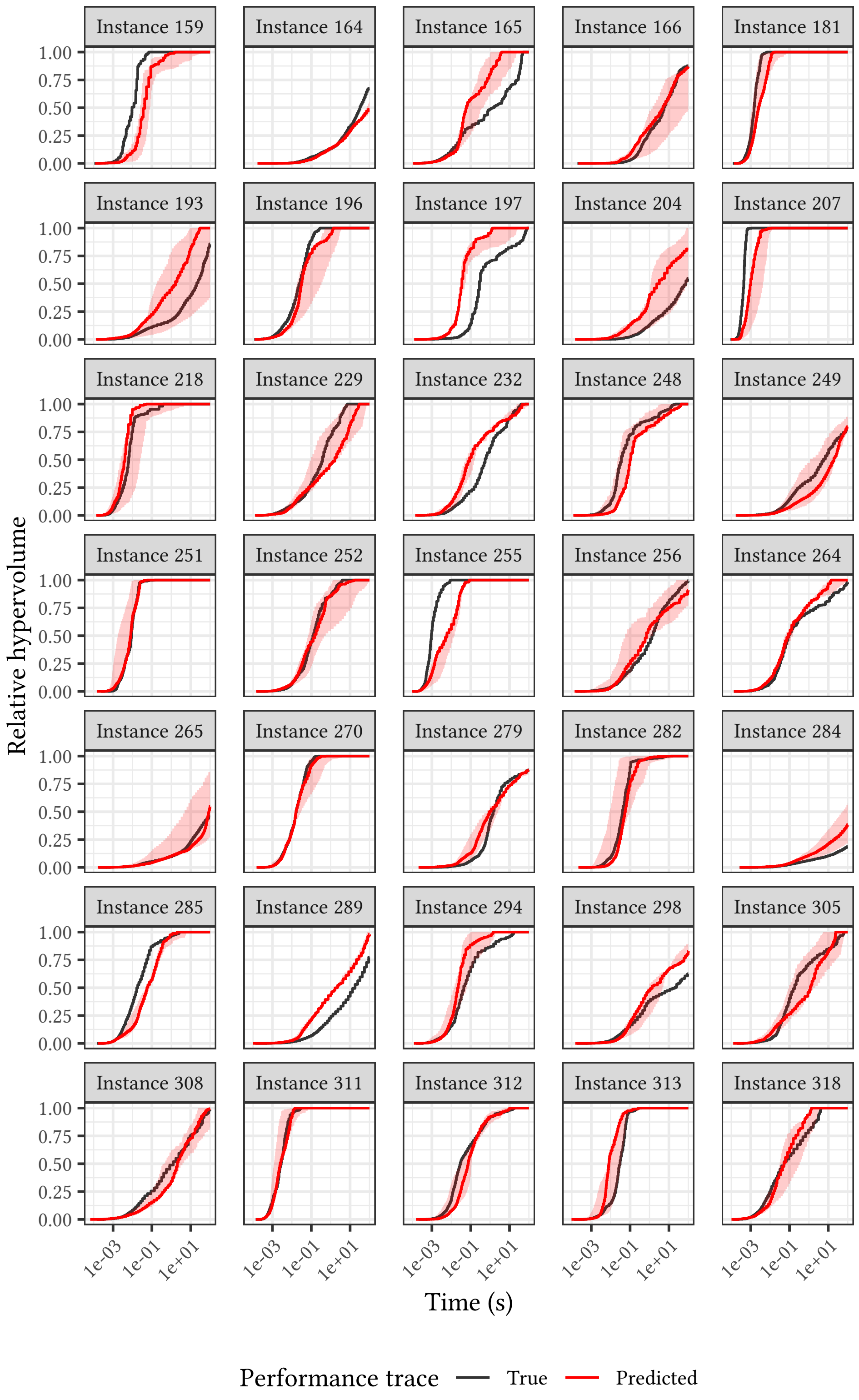


Figure A.14: Results of anytime performance prediction on the testing data set.

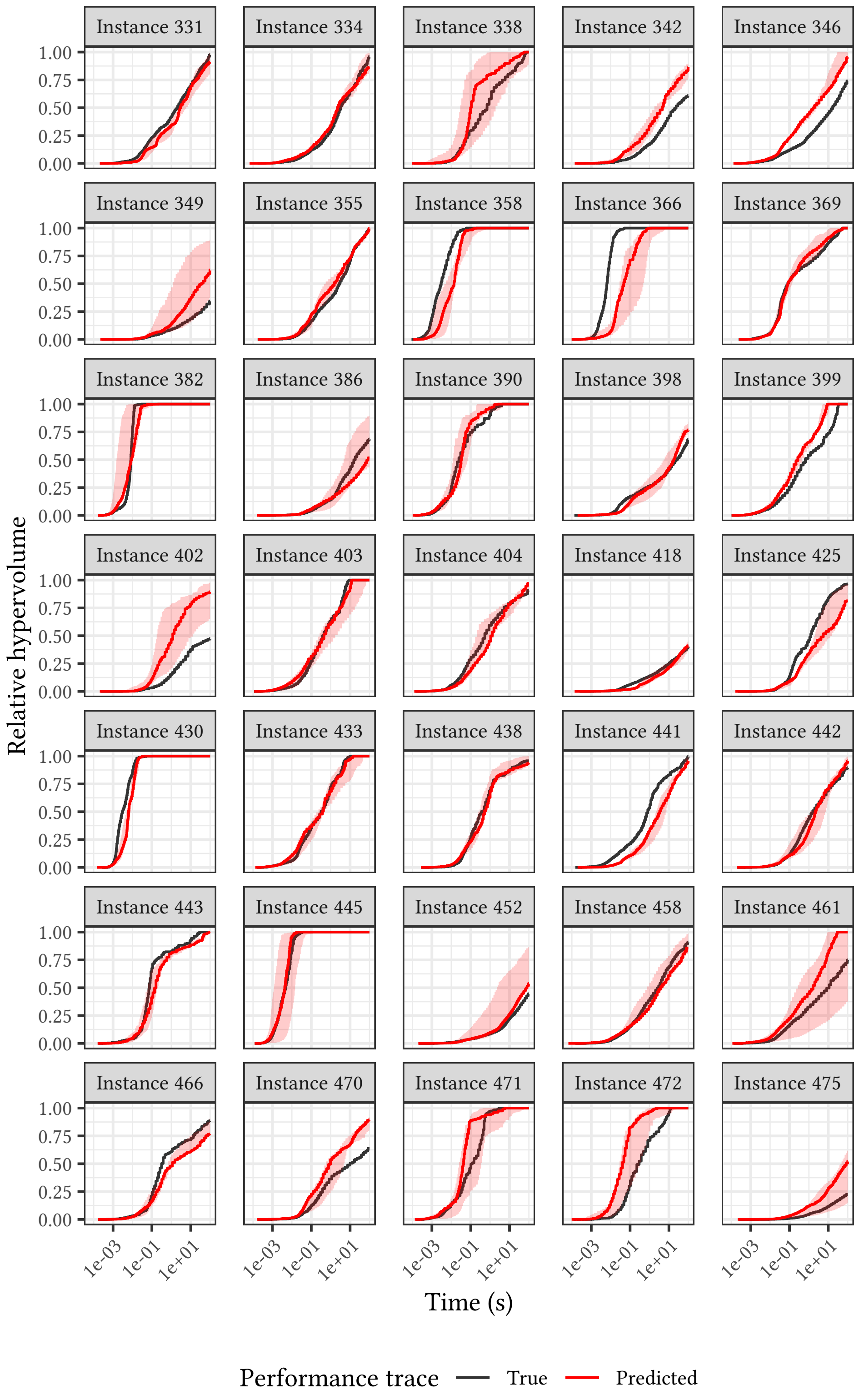


Figure A.15: Results of anytime performance prediction on the testing data set.

A.6 BHV-DP — 5 objectives

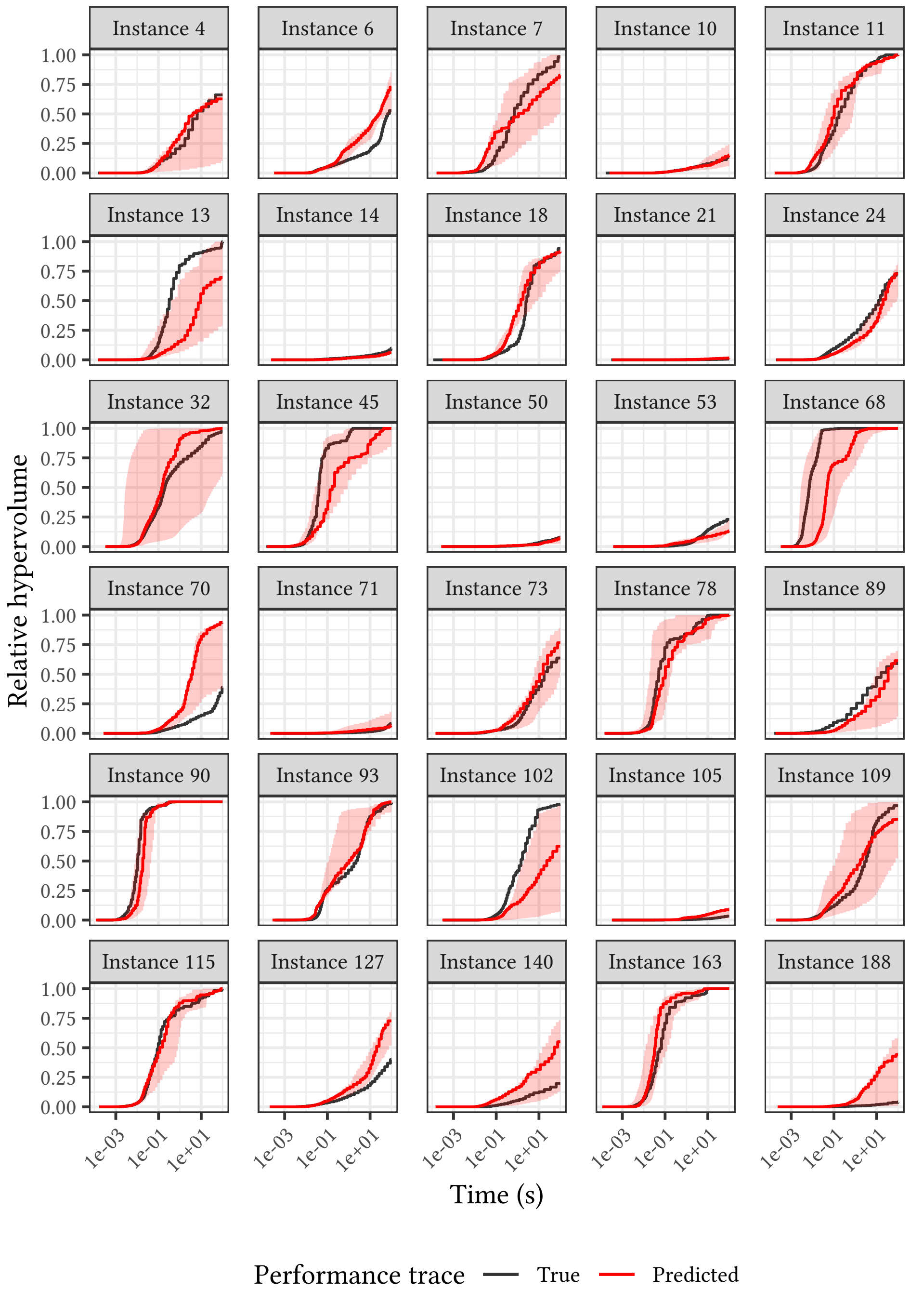


Figure A.16: Results of anytime performance prediction on the testing data set.

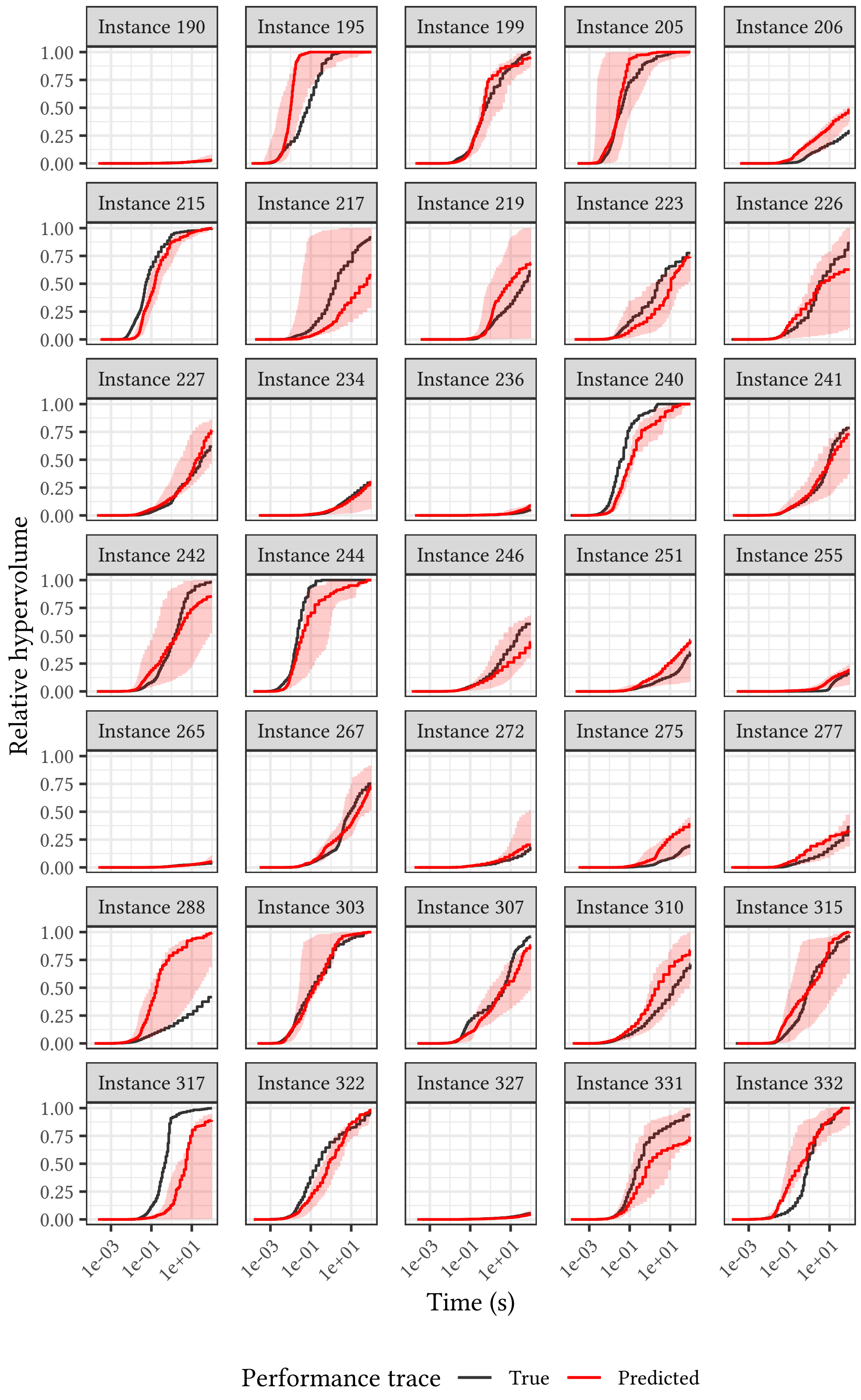


Figure A.17: Results of anytime performance prediction on the testing data set.

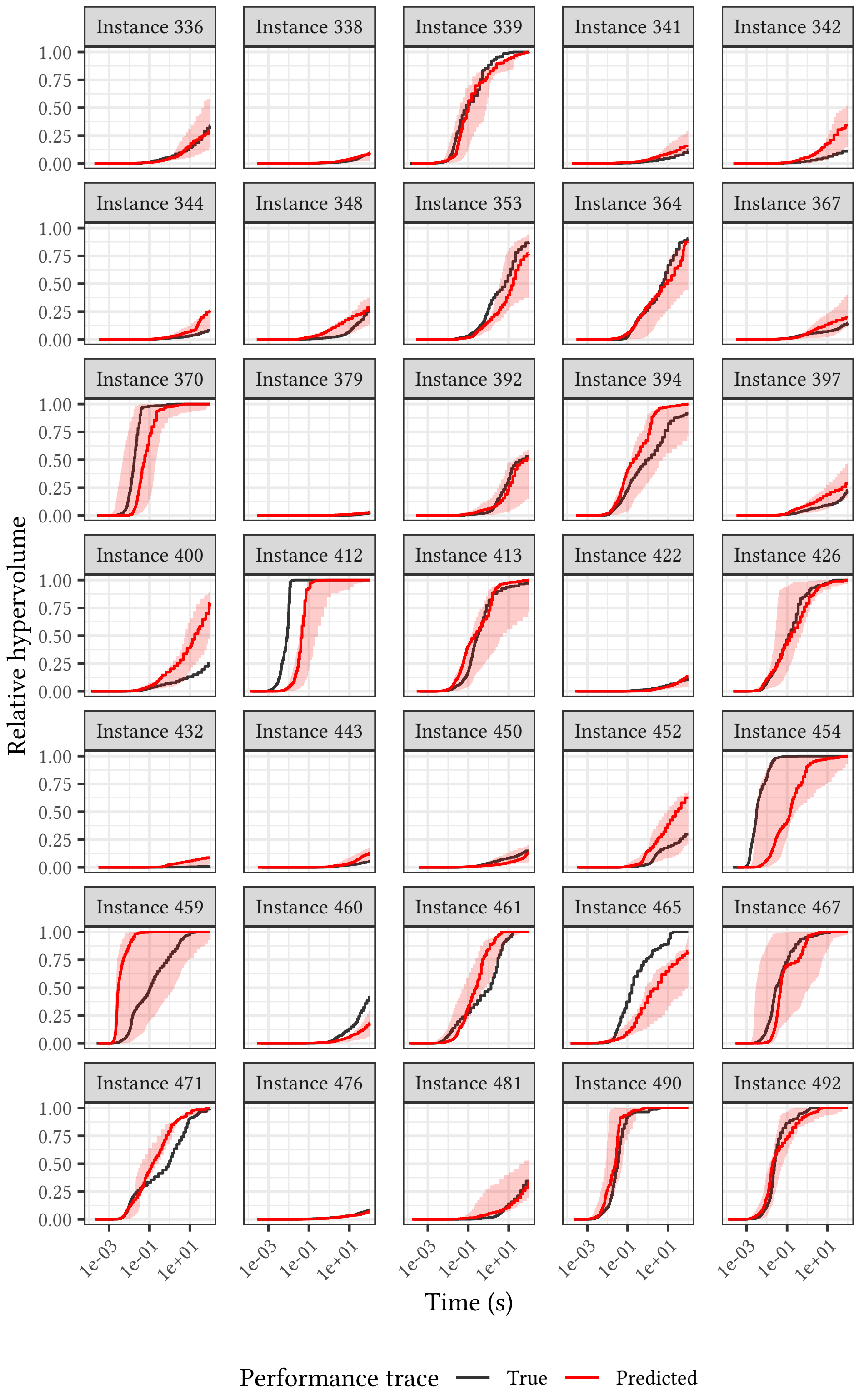


Figure A.18: Results of anytime performance prediction on the testing data set.

A.7 GEPS — 2 objectives

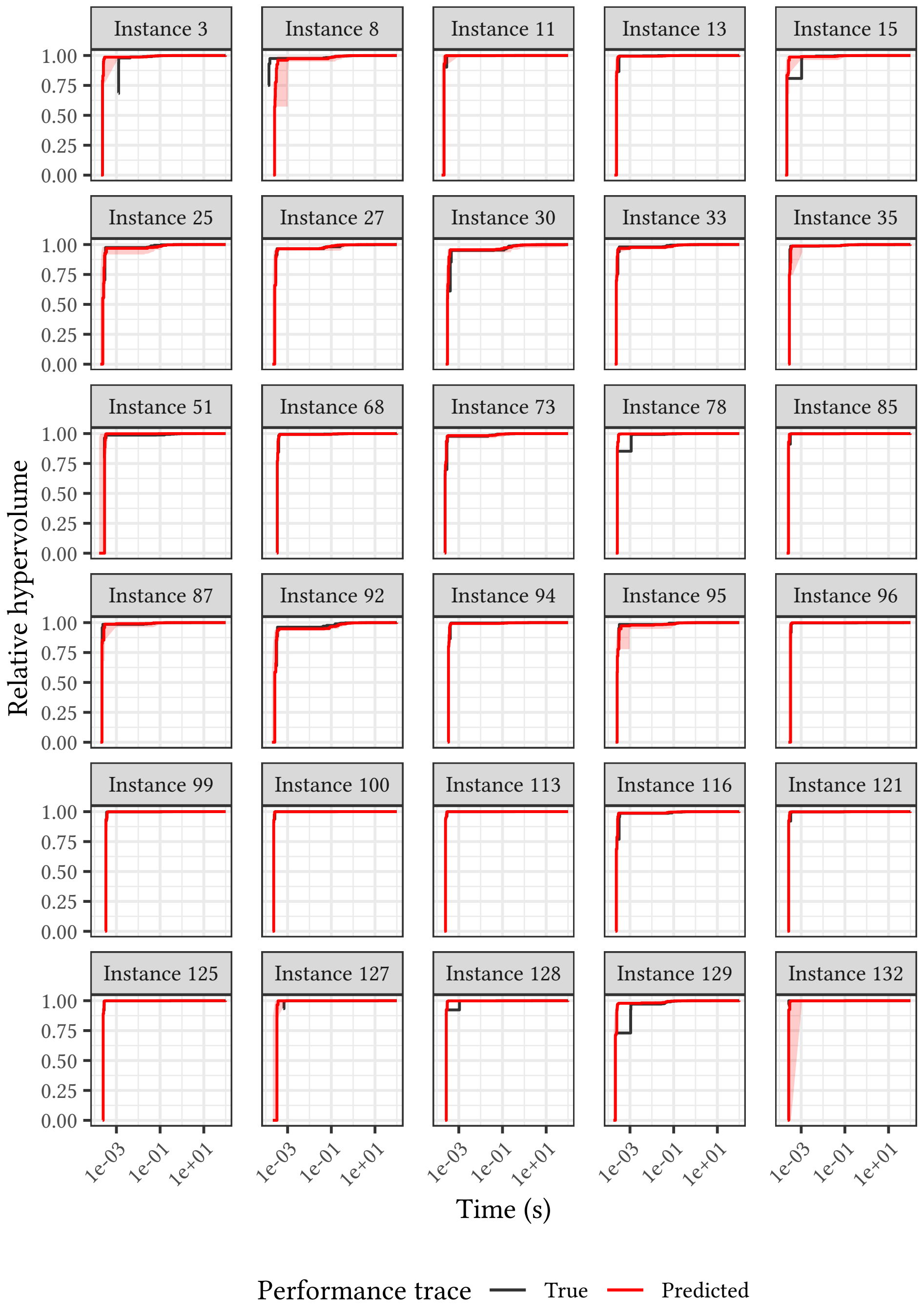


Figure A.19: Results of anytime performance prediction on the testing data set.

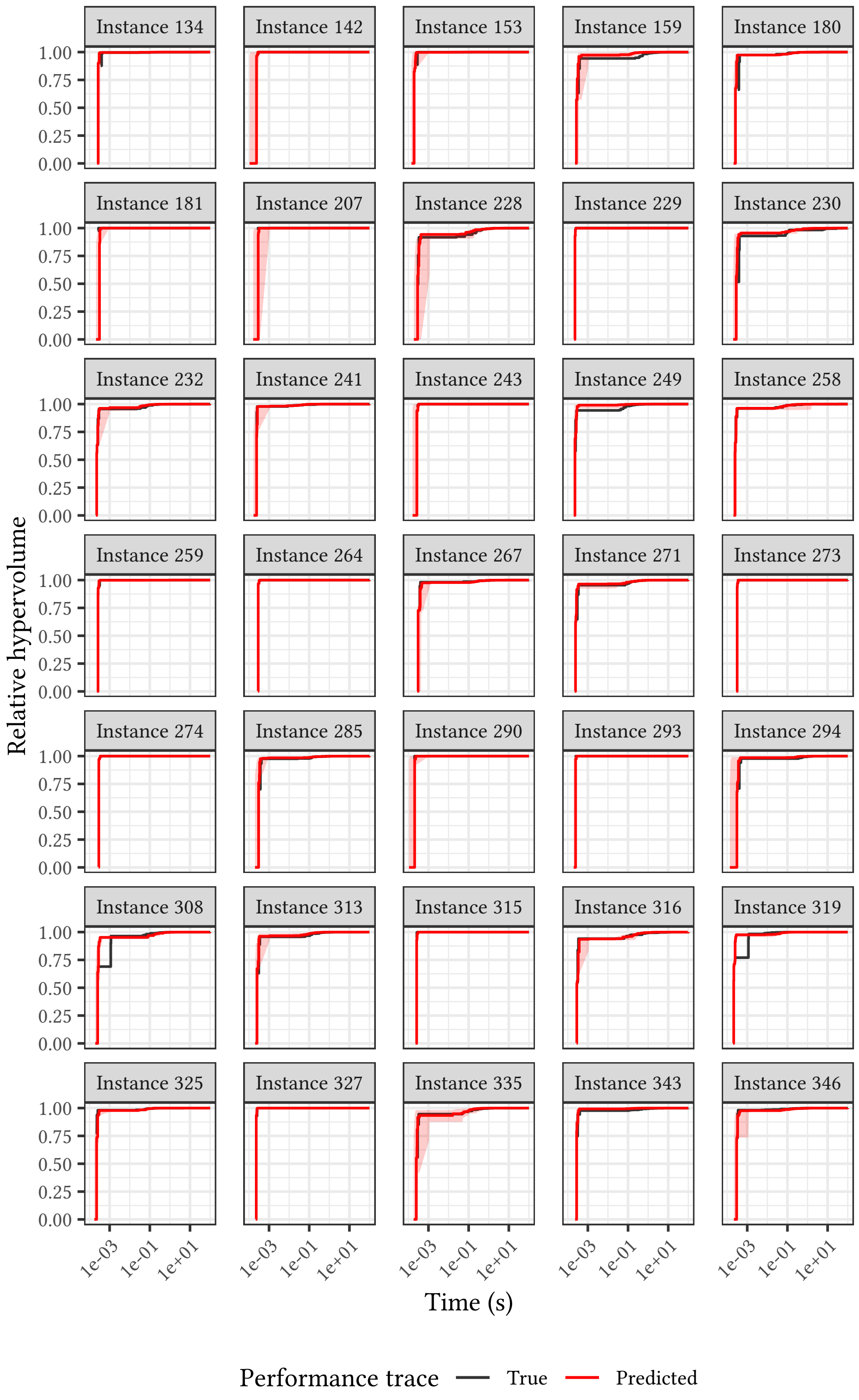


Figure A.20: Results of anytime performance prediction on the testing data set.

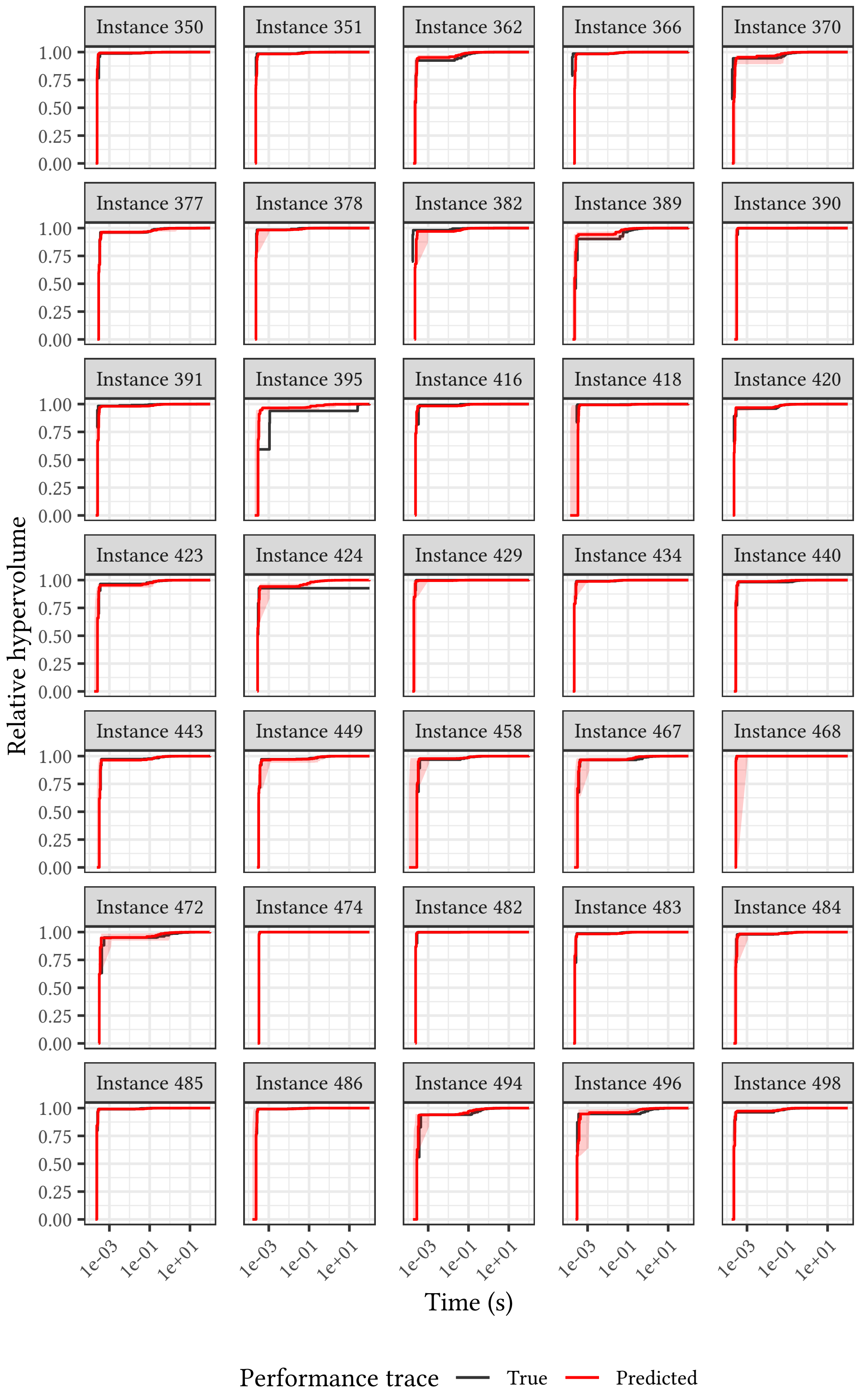


Figure A.21: Results of anytime performance prediction on the testing data set.