Advanced Algorithm Selection with Machine Learning:

Handling Large Algorithm Sets, Learning From Censored Data, and Simplifying Meta Level Decisions

---

Alexander Tornede

Defense: June 19, 2023
Version: Published

PADERBORN UNIVERSITY
Department of Computer Science
Warburger Straße 100
33098 Paderborn

Dissertation

In partial fulfillment of the requirements for the academic degree of
Doctor rerum naturalium (Dr. rer. nat.)

Advanced Algorithm Selection with Machine Learning:

Handling Large Algorithm Sets, Learning From Censored Data, and Simplifying Meta Level Decisions

Alexander Tornede

1^{st} *Reviewer*	Prof. Dr. Eyke Hüllermeier Institute of Informatics Ludwig Maximilian University of Munich
2^{nd} *Reviewer*	Prof. Dr. Axel-Cyrille Ngonga Ngomo Department of Computer Science Paderborn University
3^{rd} *Reviewer (external)*	Prof. Dr. Marius Lindauer Institute of Artificial Intelligence Leibniz University Hannover
Supervisor	Prof. Dr. Eyke Hüllermeier

Defense: June 19, 2023

Alexander Tornede
Algorithm Selection with Machine Learning:
Handling Large Algorithm Sets, Learning From Censored Data, and
Simplifying Meta Level Decisions
Dissertation, Defense: June 19, 2023
Reviewers: Prof. Dr. Eyke Hüllermeier, Prof. Dr. Axel-Cyrille Ngonga Ngomo
and Prof. Dr. Marius Lindauer
Supervisor: Prof. Dr. Eyke Hüllermeier
Department of Computer Science
Warburger Straße 100
33098 Paderborn

Abstract

There exists a plethora of algorithms for most computationally hard problems, which all have their strengths and weaknesses on different instances of said problems. Correspondingly, practitioners are constantly faced with the question: Which algorithm should be chosen for this particular problem instance to achieve strong performance? Research on algorithm selection tries to answer this question by developing decision policies, called algorithm selectors, prescribing an algorithm for a given problem instance. Most of such selectors are based on data-driven learning methods leveraging recorded evaluations of algorithms on problem instances. Although many algorithm selectors have been developed over the last few decades, selecting from large sets of algorithms, learning from censored data, and choosing an appropriate algorithm selector itself remain important practical challenges. With this thesis, we substantially improve the practical applicability of algorithm selection by suggesting advances to the underlying machine learning methods to cope with the challenges mentioned above. In particular, we demonstrate that representing algorithms in the form of feature vectors enables one to efficiently learn an algorithm selector capable of selecting from an extremely large set of algorithms. Moreover, we leverage methods from the field of survival analysis and multi-armed bandits to let algorithm selectors learn from censored data in offline and online settings. On a more abstract level, we employ ensemble learning techniques to combine multiple algorithm selectors into a single meta selector, reducing the burden of selecting an appropriate algorithm selector for a practitioner and further improving the robustness of algorithm selection. In extensive experimental evaluations on standard algorithm selection benchmarks, we demonstrate the effectiveness of our solutions. Our contributions do not only overcome some of the last challenges for democratizing algorithm selection but also have the potential to lead to impactful research on related meta algorithmic problems such as algorithm configuration and automated machine learning.

Zusammenfassung

Für die meisten rechenintensiven Probleme gibt es eine Vielzahl an Algorithmen, die alle ihre Stärken und Schwächen auf verschiedenen Instanzen der genannten Probleme haben. Dementsprechend sehen sich Anwender ständig mit der Frage konfrontiert: Welcher Algorithmus sollte für dieses spezielle Problem gewählt werden, um eine hohe Lösungsgüte zu erreichen? Die Forschung auf dem Gebiet der Algorithmenselektion versucht, diese Frage zu beantworten, indem sie Entscheidungsstrategien, so genannte Algorithmenselektoren, entwickelt, die einen Algorithmus für eine bestimmte Probleminstanz vorschlagen. Die meisten dieser Selektoren basieren auf datengetriebenen Lernmethoden, die auf Basis aufgezeichneter Evaluationen von Algorithmen auf Probleminstanzen betrieben werden. Obwohl in den letzten Jahrzehnten viele Algorithmenselektoren entwickelt wurden, bleiben die Auswahl aus großen Mengen von Algorithmen, das Lernen auf Basis zensierter Daten, und die Wahl eines geeigneten Algorithmenselektors selbst, wichtige praktische Herausforderungen. In dieser Arbeit verbessern wir die praktische Anwendbarkeit der Algorithmenauswahl erheblich, indem wir Fortschritte bei den zugrunde liegenden Methoden des maschinellen Lernens vorschlagen, um die oben genannten Herausforderungen zu bewältigen. Insbesondere zeigen wir, dass die Darstellung von Algorithmen in Form von Featurevektoren es ermöglicht, einen Algorithmenselektor effizient zu lernen, der in der Lage ist, aus einer extrem großen Menge von Algorithmen auszuwählen. Darüber hinaus nutzen wir Methoden aus dem Bereich der Überlebensanalyse und der mehrarmigen Banditen, um Algorithmenselektoren aus zensierten Daten in Offline- und Online-Settings zu lernen. Auf einer abstrakteren Ebene setzen wir Ensemble-Lerntechniken ein, um mehrere Algorithmenselektoren zu einem einzigen Meta-Selektor zu kombinieren, wodurch die Last der Auswahl eines geeigneten Algorithmus-Selektors für einen Anwender verringert und die Robustheit der Algorithenselektion weiter verbessert wird. In umfangreichen experimentellen Auswertungen mit Standard-Benchmarks zur Algorithmenselektion zeigen wir die Wirksamkeit unserer Lösungen. Unsere Beiträge überwinden nicht nur einige der letzten Herausforderungen bei der Demokratisierung von Algorithmenselektion, sondern haben auch das Potenzial, zu bedeutenden Forschungsarbeiten über verwandte meta-algorithmische Probleme wie die Algorithmenkonfiguration und das automatisierte maschinelle Lernen zu führen.

Acknowledgement

What an incredible ride the years of my Ph.D. endeavor have been! I was lucky to be supported by and work with many awesome people along the way, and I want to take this opportunity to thank them. While there certainly is a weak ordering among the different paragraphs below, most of the ordering is not meant to express any form of significance in my gratefulness. Despite great care, I will most likely have forgotten someone important in the already quite extensive list below — if you are among them, please let me thank you upfront.

First, I want to thank my Ph.D. supervisor Eyke Hüllermeier, whom I got to know when I was still a bachelor's student. You sparked the joy for data-driven methods in me and supported me already during my time as a student by enabling me to work on several projects in your group. Once close to the end of my studies, you offered me the opportunity to do a Ph.D. in your group. The time you invested in me, especially during my first year as a Ph.D. student when the Intelligent Systems group at Paderborn University was still rather small, helped me to quickly get into research, deepen my understanding of a broad spectrum of topics, and become a much better writer. Moreover, your rigorous theoretical view on topics is an incredibly useful skill, which shaped my view on topics as well as me as a researcher in general. I would like to highlight that even when you decided to accept an offer from the LMU, you continued to supervise all of us, who did not decide to move to the south of Germany with you, and ensured that everyone had sufficient funding for a reasonable amount of time.

Second, I want to thank my former office mate Marcel Wever for onboarding me so well from the very first day on. Your support enabled me to kickstart my Ph.D. Our discussions and your critical opinion on topics formed the researcher I am today. More importantly, I will never forget the fun we had in our office when trash-talking and making bad jokes. I am incredibly grateful for the time with you and for the friendship we have formed.

Third, I would like to thank Kevin Tierney for sparking my joy for artificial intelligence topics, in particular, optimization, and supporting me throughout my bachelor and master studies. Your extremely motivating and funny manner made working with you always fun, and it managed to take my head off the stress of my studies. I

am not sure whether I would have done a Ph.D. in machine learning without getting in touch with artificial intelligence first in your courses in such a great way. Thank you for being such an awesome teacher and taking me under your wing when I was still a bachelor's student.

Fourth, I want to thank all former members of the Intelligent Systems group at Paderborn University. Thank you, for flawlessly integrating me as a new team member. I will always remember our discussions during our sometimes rather extensive coffee and lunch breaks and, in particular, our Eldritch Horror sessions on weekends! In particular, I would like to thank Felix Mohr, who onboarded me together with Marcel and also guided me during my first year as a Ph.D. student before he left for Columbia to become a professor. Moreover, I would like to thank Elisabeth Lengeling, our secretary, who has gone above and beyond every day to make sure that everyone of us could focus on their research. You took extremely much off our shoulders in terms of organizational work but also kept us mentally sane with your kind words and consistent supply of sweets. Similarly, I would like to thank Theodor Lettmann, who always had an open door for everyone. Let it be research technicalities, organizational matters, or personal problems — you always listened and offered valuable advice. Thank you, for always being there, Theo. At the same time, I would like to thank the "ZM2 gang" — the part of the Intelligent Systems group, which relocated to the new ZM2 building after Eyke left Paderborn. The lunch breaks, social evenings, and discussions we had made my last months in Paderborn much more enjoyable. Moreover, I would like to thank Helena Graf, in particular, whom I truly got to know during the last few years. I do not want to miss the truly meaningful friendship we have formed. I am very grateful for the time and conversations we shared and your emotional support during the last stages of this work.

Fifth, I want to thank the CRC 901 for funding most of my time as Ph.D. student and, in particular, Friedhelm Meyer auf der Heide for becoming my "Fachvorgesetzter" after Eyke left. During our first conversation, you offered me a safe harbor in case any problems would arise during my remaining time, but also left me the freedom to finish this thesis. Moreover, I want to thank Ulf-Peter Schröder and Marion Hucke for their support with any organizational questions regarding the CRC and my contracts. On the same occasion, I want to thank the SICP and, in particular, Henning Wachsmuth and Stefan Sauer for offering me an office at the ZM2, after Eyke left for Munich. You made sure that the rest of the group could stay together, which helped us immensely in keeping in touch with each other and helping each

other with our research. I would also like to thank Sonja Saage for helping out with any organizational issues, which came with our move to the ZM2.

Sixth, I want to thank all of my incredible and awesome co-authors for working with me on our various meta algorithmic projects. Your perspectives and opinions helped me shape my view on the topic and made me both a much better writer and researcher. It's been a blast with you! Similarly, I want to thank Stefan Werner and Lukas Gehring, who worked for me as student assistants during my time as a Ph.D. student and took some cumbersome work off my shoulders.

Seventh, I want to thank Karlson Pfannschmidt, again Marcel Wever, and my wife Tanja Tornede for proofreading my thesis, and Viktor Bengs for proofreading the more theoretical parts. Your suggestions and critique made this a much better work. In addition, Karlson was always of great help when I had any thesis design questions. Similarly, I want to thank Tanja and Helena for their help in designing most of the figures presented in this thesis. I also want to thank all members of my Ph.D. committee for agreeing to be part of it and, in particular, the reviewers for taking the time to review this thesis. I know that your to-do lists are only ever-growing and that your time is precious. Thus, I really appreciate that you take the time to dive into the work, which I have spent my time with during the last years.

Eighth, and starting with a more personal matter, I want to thank my family for their trust in me. In particular, I want to thank my mother for raising me to be a curious person and thus offering me the opportunity to study in the first place. Furthermore, I want to thank my aunt and my two cousins with their families for being with me during the last few years. Your support helped me through some hard times, and our events were a source of joy during good times. I would also like to thank the family Tornede, which I have married into, for welcoming me so warmheartedly, in particular, my father-in-law, who always has our back.

Last, but not least and most importantly, I want to thank my beloved gorgeous wife, Tanja Tornede, for her constant, unbreakable support even in the darkest times. You have always had my back and helped me get back up when I struggled. The fun we shared always got my mind off of work, but at the same time, I cannot express how much I love to tech talk with you about our research. As a side note, this thesis certainly would have looked different without your help in designing figures, fixing LaTeX code, and keeping me sane during the last sprint. I am incredibly grateful for the opportunity to share our lives with each other and light up the fire together. I would by far not be the person I am today without you. I love you ♡.

Contents

1 Introduction

The world we live in has become increasingly complex along various dimensions, and this trend seems to be increasing in speed. Overcoming particular challenges, such as reliable and large-scale goods logistics, sometimes incurs new challenges, for which other solutions are sought. Often such challenges are extremely hard to solve optimally for humans. Correspondingly, more and more such hard problems are nowadays solved by algorithms on computers. As a computer scientist, the problem of Boolean satisfiability (SAT) will immediately come to one's mind, which is about determining whether there is a satisfying interpretation of a Boolean formula. Despite its age, SAT has quite some practical relevance as a (sub-)problem in fields such as model checking, planning or bioinformatics [Mar08]. More immediate, consider, for example, the problem of stacking containers at large logistics ports in a way that minimizes crane movements when vessels have to be loaded, known as the container premarshalling problem (CPMP) [TM15], or the problem of routing a mailperson delivering packages in a fuel-saving way between customers — a form of the vehicle routing problem (VRP) [EVR09]. In fact, most of such combinatorial optimization problems are NP-hard and thus even hard to solve for modern computers in general. As a consequence, there often exists a plethora of different algorithms solving the same problem with different strengths and weaknesses, which are often designed to perform well on different edge cases. This is not very surprising and quite in line with theoretical results proving that there is "no free lunch", i.e. that no algorithm uniformly dominates all others on a problem under certain assumptions [WM97]. This phenomenon is sometimes denoted as *performance complementarity* among the algorithms [Ker+19].

While it is generally positive that the set of applicable algorithms is often quite extensive, given a certain algorithmic problem, this situation imposes a new problem on the practitioner: Which of these algorithms should be chosen in what situation? Or in other words, given a specific instance of such a problem, e.g., a certain set of containers with departure times (CPMP) or a certain set of package and customer locations (VRP), which algorithm generates the most suitable solution? This question was first formalized by Rice [Ric76] coining the algorithm selection (AS) problem

in the last century, quantifying suitability in terms of a function mapping from the instance space and algorithm set (via detours) to the real numbers.

Since then, AS has received considerable attention in the scientific world featuring a plethora of publications (nicely summarized in two surveys [Ker+19; Kot16]) elaborating on the problem and/or suggesting different algorithm selection solutions, called algorithm selectors. The field comprises several competitions [LRK19], scientific events, and communities [COSEAL; AutoML-Conf; MTL; AutoML-FS]. Moreover, AS was generalized towards other related meta algorithmic problems such as algorithm configuration (AC) [Sch+22] or automated machine learning (AutoML) [HKV19] — some of which have become extremely popular on their own.

The vast majority of existing approaches to AS are data-driven and based on machine learning. In order to make good algorithm recommendations, approaches learn from existing data featuring algorithm runs on prior problem instances. Such instances are represented in the form of characteristics describing relevant properties, known as instance features, in order to generalize across them [Ric76; Ker+19].

The goal of this thesis is to advance AS by drastically improving the core machine learning elements used in modern algorithm selectors to be more suited for the challenges arising from the data in varying AS settings. In particular, we design algorithm selectors powered by learning algorithms leveraging solutions to the aforementioned challenges. By doing so, we aim to provide algorithm selectors with a better selection performance under challenging conditions than the hitherto state of the art and to make it faster and less cumbersome for practitioners to apply algorithm selection tailored to their needs. We strongly believe that this thesis represents a significant step forward for the applicability of algorithm selection and that it has the potential to impact a wide range of applications, including machine learning and optimization as we detail in Section 1.2.

The remainder of this chapter is organized as follows: Section 1.1 gives an overview of the contributions presented in this thesis, including corresponding preceding publications made during my time as a Ph.D. student, and also explains the overall thesis structure. Section 1.2 aims at detailing the potential impact of this thesis. Section 1.3 elaborates on the scope of this thesis and, in particular, what elements of the AS problem this thesis is focused on and which elements are only sparsely discussed. In order to facilitate the reading process, Section 1.4 contains some tips and tricks, which hopefully help to digest this thesis more easily. The introduction is

closed by Section 1.5, where I give credit to the co-authors of my publications, on which this thesis is partly based.

As common, throughout this thesis, I will use the scientific ”we” instead of ”I” except for very few parts where it is important to avoid ambiguities in the credit assignments to my co-authors.

1.1 Content, Contributions and Preceding Publications

The following serves as an overview of both the remainder of this thesis and the contributions made by the remaining chapters of this thesis. Figure 1.1 tries to visually capture, in particular, the methodological contributions described in the following.

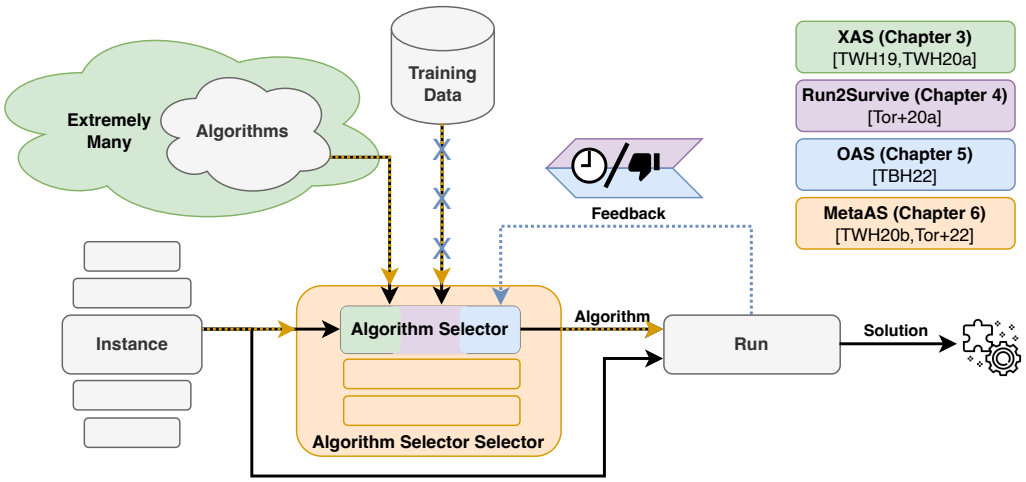

Fig. 1.1: Highlight of the contributions of this thesis affecting various components of AS, color-coded, such that each main chapter has its own color.

Chapter 2: This chapter contains prerequisites, which are often assumed to be known in the AS literature. In particular, we formally define several AS problem variants — some of which have been developed as part of work put into this thesis — in a unified fashion, compare them to each other and distinguish the AS problem from related ones such as AutoML. Moreover, we present a thorough taxonomy of existing algorithm selection approaches, formally defining them within a unified framework, and discuss the strengths and weaknesses of the corresponding approaches. Similarly, we describe existing loss functions for AS, once again including a discussion of strengths

and weaknesses. We close the background chapter by discussing desired properties of instance features and reviewing methods for generating such features.

Chapter 3: This chapter deals with a variant of the algorithm selection problem, where the number of algorithms to choose from grows extremely large such that existing algorithm selectors reach their limits. We dub this the extreme algorithm selection (XAS) problem and discuss several implications coming with the increase in the number of algorithms, most importantly, the limitations of standard approaches to algorithm selection in such cases and the corresponding causes. To alleviate these limitations, we suggest not only representing instances but also algorithms by features yielding a dyadic feature representation that allows learning a single joint model across all algorithms. In order to show that our proposed solution does indeed alleviate the problems, we design an XAS benchmark scenario and perform an extensive experimental evaluation. Parts of this chapter have been published in [TWH19; TWH20a].

Chapter 4: In this chapter, we investigate the implications of the common procedure of executing algorithms under tight time constraints, so-called cutoffs, after which the algorithm is forcefully terminated to avoid running them extremely long. As a consequence of this procedure, targets of datapoints arising from such algorithm runs are only of limited use to standard learning algorithms as they constitute a right-open interval, called right-censored datapoint, instead of a precise numeric value. We elaborate on the consequences arising from this observation and on the disadvantages of existing methods to cope with such censored datapoints. Furthermore, we propose the use of survival analysis methods together with a decision-theoretic decision rule to learn risk-averse algorithm selectors, which inherently can learn from censored data. Our experimental study reveals that our proposed strategy yields a new state-of-the-art algorithm selector called Run2Survive. Parts of this chapter have been published in [Tor+20a].

Chapter 5: In contrast to the previously discussed problem of censored data in the offline AS setting, this chapter deals with solving the same problem in an online manner, i.e. the online AS setting. Although this transition might seem easy at first glance, the online setting requires completely different approaches since, for most of the standard survival analysis approaches, it is not easily possible to update them incrementally. Instead, we revisit well-known contextual bandit approaches and discuss their suitability for dealing with the online algorithm

selection (OAS) problem and adapt them in a theoretically grounded way towards runtime-oriented losses under the assumption of partially censored data with a time- and space-complexity independent of the time horizon. To empirically validate our proposed approaches, we perform an extensive experimental evaluation and show that we can improve upon existing OAS approaches in terms of performance while featuring a better time- and space-complexity. Parts of this chapter have been published in [TBH22].

Chapter 6: Taking a step back, we consider the AS problem from a meta level motivated by the idea that the performance complementarity among algorithms can also be observed among the algorithm selectors. Correspondingly, we tackle the problem of selecting between algorithm selectors for a given instance. To this end, we present several solutions, including meta learning an algorithm selector and ensembling multiple selectors. In an extensive experimental evaluation, we show that the meta algorithm selection (MetaAS) problem can be solved efficiently and that solutions can provide remarkable improvements in performance, often significantly better than the hitherto state of the art. Parts of this chapter have been published in [TWH20b; Tor+22].

Chapter 7: We close this thesis by discussing its content in a larger context and elaborate on possible paths for future work in a broader context, connecting several of the topics discussed throughout this thesis.

1.2 Potential Impact of This Thesis

The contributions made within this thesis have the potential for a large impact along two dimensions: practicability of AS and research on related meta algorithmic problems.

1.2.1 Practicability of Algorithm Selection

Since no algorithm dominates all others on all instances of a problem, for many problems, there exists a plethora of algorithms that are tailored to perform well in very specific cases, such as instances with certain properties. Unfortunately, most algorithm selectors can only effectively choose from a handful to tens of algorithms

due to the design of their underlying machine learning model. As a consequence, a practitioner has to pre-select some algorithms, which the algorithm selector can then choose from. Naturally, such a pre-selection is not only a tedious task but also bears the danger of making detrimental mistakes as the practitioner has to anticipate which algorithms might be more useful than others. Our work on XAS removes this burden from the practitioner as we develop algorithm selectors capable of choosing from an extremely large set of algorithms such that no pre-selection has to be performed. Correspondingly, the practitioner can focus on other important design decisions, such as designing good instance features (cf. Section 2.5), enabling a faster and less error-prone algorithm selection application.

Similarly, most existing algorithm selectors can only deal with censored data to a certain degree such that, ideally, training data for algorithm selectors should not contain any censoring. Consequently, a practitioner wanting to apply AS has to ensure high training data density in the sense that, in the best case, each algorithm has been evaluated on many training instances. This is not only very time-consuming and often practically unachievable due to extremely long algorithm runtimes but, in the spirit of green meta algorithmic topics [Tor+21a], is also harmful to the environment due to unnecessary evaluations and produced CO_2. Our contributions in Chapter 4 and Chapter 5 allow for a faster algorithm selection system setup as we enable effective training of algorithm selectors based on censored data both in an offline and online scenario.

Lastly, whenever a practitioner aims at using AS, they are faced with the question of which algorithm selector to employ for their system. Although this question does not have to be answered for each instance necessarily, it has to be answered at least once when setting up the overall AS workflow for their use case. Once again, this decision is complex as algorithm selectors are complementary, which make them more suited for some settings than for others. Our work on meta algorithm selection presented in Chapter 6 reduces this burden of the practitioner by demonstrating that ensembling algorithm selectors yields state-of-the-art algorithm selection performance. As such, they do no longer have to choose an algorithm selector, but can leverage our meta algorithm selector, which ensembles the state of the art in algorithm selection.

Overall, the contributions of this thesis drastically improve the AS workflow from a practitioner's point of view. We enable true end-to-end algorithm selection, removing possibilities for errors, improving the setup time of algorithm selection systems, and thus substantially improve the robustness of AS.

1.2.2 Research on Related Meta Algorithmic Problems

Some of the contributions of this thesis have the potential to improve the practicability of solutions to other related meta algorithmic problems in a similar manner. In particular, properly dealing with censored data through dedicated methods instead of rather naive solutions and selecting among or ensembling meta algorithmic approaches can be, and in parts have already been, generalized to other problems, such as algorithm configuration or automated machine learning. Moreover, combinations of the contributions made within this thesis open up new interesting AS problem variants as we detail in Section 7.2.

1.3 Scope of This Thesis

As AS is a very large field, even long scientific manuscripts such as a thesis like this have to focus on certain parts and rather neglect others. This thesis is mostly centered around machine learning based meta algorithmic approaches to algorithm selection. In particular, we focus on the machine learning techniques at the core of most algorithm selectors and try to elaborate on weaknesses of existing solutions. To alleviate these, we propose mitigation techniques in different settings of the AS problem. We largely ignore the problem of instance feature design/generation in the technical chapters of this thesis and, instead, give an overview as part of the background (Section 2.5.2) and simply assume instance features to be available. Furthermore, we largely neglect other components which can often be found in algorithm selectors, such as feature selection. Instead, we focus on the design and evaluation of the aforementioned machine learning model at the core of an algorithm selector. Correspondingly, we use very simple feature preprocessing across all compared selectors in the different evaluation sections (they may vary by evaluation section, though), use the standard configuration of most selectors, and do not perform any presolving. We are aware that these are rather restrictive assumptions, but they allow us to focus on the core component and rule out many possible confounding factors when evaluating our proposed approaches.

1.4 How to Read This Thesis

To alleviate some of the hassles that often come with reading documents as long as this thesis, we suggest certain reading strategies based on the reader's knowledge in AS followed by some general tips.

Beginner: As a beginner, we recommend reading the complete background chapter before advancing to the technical chapters to get familiar with the problem and some assumptions, which are rather hidden in the AS literature.

Advanced: As an advanced researcher in AS, the reader can most likely skip large parts of the background chapter. However, we recommend skimming the problem definitions (Section 2.1.1, Section 2.1.2 and Section 2.1.3) to get accustomed with the notation. For the same reason, we recommend skimming our taxonomy of algorithm selection solutions (Section 2.3).

The technical chapters, i.e. Chapters 3–6 can in principle be read independently of each other and in a rather arbitrary order. However, we recommend reading Chapter 4 before Chapter 5 to support a better understanding. Nevertheless, when reading the chapters in a different order, some back pointers might not be easily understandable.

A list of used abbreviations can be found in Chapter A.5.3. Furthermore, all abbreviations are hyperlinks such that clicking on the abbreviations brings the reader to their meaning. Similarly, all notation frequently used in formulas can be found in Table A.5.3 for reference. Symbols, which are only used once or twice within the paragraph where they are defined, are left out to avoid cluttering the list. Similarly, the list only contains atomic symbols, i.e. composite symbols constructed from those atomic ones are not listed.

The appendix contains details on the experimental evaluations presented in the corresponding chapters featuring reproducibility information. It also contains the proofs for the theoretical results presented in Chapter 5.

1.5 Co-Author Contribution Statement

As noted at the start of the corresponding chapters, some partial content of chapters of this thesis has been published at workshops, conferences, and journals during my time as a Ph.D. student. I was fortunate enough to collaborate with fantastic colleagues on all of my papers and thus would like to note my particular contribution to each of these publications in the context of all authors' contributions in the following. The papers are grouped by the respective chapters, which are based on them.

Chapter 3

[TWH19] Eyke Hüllermeier motivated this work and I took the lead in writing. An original version of the manuscript was written by myself and Eyke Hüllermeier and later jointly improved by all authors. The actual evaluation was implemented by me, although the underlying dyad ranking model was implemented by students as part of a group project.

[TWH20a] I both motivated this work and took the lead in writing. An original version of the manuscript was written by myself and later jointly improved by all authors. The actual evaluation was implemented by me, although the underlying dyad ranking model was implemented by students as part of a group project and the Alors approach by a student assistant. Furthermore, Marcel Wever wrote code for the generation of the result tables and the statistical tests.

Chapter 4

[Tor+20a] The idea of employing Survival Analysis for AS was motivated by Eyke Hüllermeier, who also contributed the idea of the expected PAR10 minimization within the survival analysis framework. After he pitched the idea to me, I took the lead. An original version of the manuscript was written by Stefan Werner and me, who wrote parts of the section on the experimental evaluation, and

later jointly improved by all authors. The evaluation was jointly implemented by Stefan Werner and me, who worked as a student assistant for me.

Chapter 5

[TBH22] Eyke Hüllermeier and me jointly motivated this work, which was then lead by me. An original version of the manuscript was written by Viktor Bengs and me, who initially derived and described most of the theoretical results shown in the paper, which were later jointly refined by the two of us. In particular, the theoretical details given in Section A.4 were derived and described by Viktor Bengs and jointly refined. I further refined the description of these derivations for the appendix of this thesis. The idea to incorporate imputation through the Buckley-James estimator into the revisited Thompson sampling was proposed by me and later formally integrated into the manuscript by Viktor Bengs. The runtime modeling was proposed by me and jointly refined. The complete document was refined jointly by all authors. The evaluation was solely implemented by me.

Chapter 6

[TWH20b] This work was motivated by me, and I also took the lead in writing. An original version of the manuscript was written by myself and later jointly improved by all authors. The actual evaluation was implemented by me. Marvel Wever wrote code for the generation of the result tables.

[Tor+22] This work was motivated by me, and I also took the lead in writing. An original version of the manuscript was written by myself and later jointly improved by all authors. In particular, Eyke Hüllermeier made a substantial amount of suggestions for improvements. Most figures were designed by Tanja Tornede. The implementation of the evaluation was performed jointly by Lukas Gehring and me, who worked for me as a student assistant.

1.6 Additional Publications

Except for the papers mentioned above, I published several other papers, which are at least loosely related to AS or related concepts, but which did not result in one of the main chapters of this thesis. Together with several co-authors, I published a total of 21 papers during my time as a Ph.D. student. A complete list can be found before the final bibliography at the end of this thesis in Chapter A.5.3.

2 Background

As the thesis title suggests, the central topic of this thesis is algorithm selection; in particular, we present our work on problem extensions and novel methods. Before elaborating on each of our contributions in more detail in the following chapters, we introduce some background. To this end, we will first formally introduce three different versions of the algorithm selection (AS) problem, which we tackled as part of this thesis: The offline AS problem (Section 2.1.1), the online AS problem (Section 2.1.2) and the meta AS problem (Section 2.1.3). Right after the definition, we put them into context of each other (Section 2.1.4). Secondly, we distinguish the core AS problem from related problems in Section 2.2, followed by a taxonomy of common AS solutions (Section 2.3) where we elaborate on the working principles of most AS solutions. Furthermore, in Section 2.4, we elaborate on loss functions in the context of AS before explaining the concept of instance features in Section 2.5. Since most evaluations found in this thesis are based on the standard AS benchmark called algorithm selection library (ASlib) [Bis+16], we close this chapter with Section 2.6 by giving a short introduction to the benchmark itself.

2.1 Algorithm Selection Problem Variants

Roughly speaking, the algorithm selection problem formalizes the task of selecting an algorithm from a set of candidate algorithms, also called portfolio, most suitable for solving a given task. Here, suitability is often quantified in terms of a specific measure rating how good (or bad) a certain algorithm performs on a given task. Such tasks are often present in the form of an instance of some algorithmic problem, such as a formula of a Boolean satisfiability problem.

In the following, we will introduce and formalize three variants of the problem, which will be tackled within this thesis. The first variant (Section 2.1.1) is known as the de-facto standard version, which as originally introduced by Rice [Ric76]. Here, one is tasked with creating a decision rule to select an algorithm for a given instance, often called *algorithm selector*, in an offline phase before actual instances

have to be solved. After this phase, the approach cannot be changed. The second variant (Section 2.1.2) relaxes exactly the previously mentioned assumptions of a prior offline phase and the immutability of the approach. Instead, one is tasked with designing an algorithm selector, which can adapt itself after each instance depending on how the selected algorithm performs. The last variant (Section 2.1.3) is concerned with selecting among algorithm selectors themselves instead of the algorithms and thus works on the meta level.

2.1.1 The Instance-Specific Offline Algorithm Selection Problem

When the literature speaks about the (standard) algorithm selection problem, it mostly refers to the instance-specific offline algorithm selection problem, which is the most basic form of the AS problem and was first introduced by Rice [Ric76]. As mentioned at the beginning of this work, it is motivated by the phenomenon of performance complementarity [Ker+19] according to which different algorithms perform differently well on different tasks or instances and hence, choosing among the algorithms for a given instance can yield better solutions. We present solutions to this problem variant with the focus on different assumptions in Chapter 3 and Chapter 4.

2.1.1.1 Formal Problem Definition

More formally, the *instance-specific* offline AS problem is composed of an *instance space* $\mathcal{I}$ of some algorithmic problem[1], a *set of algorithms* $\mathcal{A}$ which can solve instances from the instance space, and a costly-to-evaluate *loss function*

$$l : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R} \tag{2.1}$$

indicating how well the given algorithm performs on the given instance. For simplicity, we assume that l maps to $\mathbb{R}$, although it could in principle also map to other spaces, e.g. a multi-dimensional vector space for multi-criteria AS settings [BT18].

[1] Examples include the boolean satisfiability (SAT) problem, traveling salesperson (TSP) problem, constraint satisfaction problem (CSP) or even just a machine learning problem such as classification.

The goal underlying the AS problem then is to find a selector

$$s : \mathcal{I} \longrightarrow \mathcal{A} \ , \tag{2.2}$$

which, given an instance $i \in \mathcal{I}$, preferably selects the optimal algorithm from the set of $\mathcal{A}$, also called portfolio, according to a loss function l. Hence, we seek to find the optimal instance-specific algorithm selector s^*, also called *oracle* or virtual best solver (VBS), defined as

$$s^*(i) \in \arg\min_{a \in \mathcal{A}} \mathbb{E}\left[l(i, a)\right] \tag{2.3}$$

for all problem instances $i \in \mathcal{I}$. The expectation operator in Equation 2.3 accounts for any potential randomness in the application of the algorithm making $l(i, a)$ a random variable. Corresponding randomness can be caused, for example, by an element of the algorithm itself, which might leverage randomization such as a random initialization or randomized heuristic.

To construct such an algorithm selector, we are provided training instances $\mathcal{I}_D \subset \mathcal{I}$ and corresponding evaluations of $l(\cdot, \cdot)$ for some, but often not all of the candidate algorithms $\mathcal{A}$. Furthermore, in order to generalize across the instance space, we assume that there exists a d-dimensional feature representation for each instance, which can be computed using an instance feature map/function $f : \mathcal{I} \longrightarrow \mathbb{R}^d$. For the remainder of this work, we will denote an instance by i and its corresponding feature representation by $\boldsymbol{f}_i$.

For the SAT problem, examples of such features include the number of clauses or the number of variables. In general, to perform reasonable instance-specific algorithm selections, such features should be correlated with the performance of the algorithms on the corresponding instance. In practice, designing such features is cumbersome and a research problem on its own, as our discussion including an overview of relevant literature in Section 2.5 shows.

2.1.2 The Online Instance-Specific Algorithm Selection Problem

The online instance-specific algorithm selection (online algorithm selection (OAS)) problem[2] generalizes the AS problem (Section 2.1.1) to the online case in the sense

[2]We would like to note that while this is our understanding and formalization of the problem, the term online algorithm selection is ambiguously understood in the community as we detail in Section 5.6

that (1) no training instances $\mathcal{I}_D$ are available and (2) the selector can be updated after each prediction / selection. Hence, the selector s has to be constructed and updated in an *online* fashion. For this purpose, it receives feedback after each selection. We present solutions to this problem variant in Chapter 5.

2.1.2.1 Formal Problem Definition

More formally, the OAS problem is an iterative decision making problem, which, just like the AS problem, consists of a problem instance space $\mathcal{I}$ and a set of candidate algorithms $\mathcal{A}$, which can solve such instances. As depicted in Figure 2.1, the problem is conducted over several rounds, i.e. timesteps, such that at each timestep an instance $i_t \in \mathcal{I}$ arrives and an algorithm $a_t \in \mathcal{A}$ has to be selected to solve the instance. This selection is performed by an *online algorithm selector*

$$s_{online} : \mathcal{H} \times \mathcal{I} \longrightarrow \mathcal{A} \ , \tag{2.4}$$

such that $a_t = s_{online}(h_t, i_t)$ where $h_t = \{(i_k, a_k, l_k)\}_{k=1}^{t-1} \in \mathcal{H}$ denotes the history of the selection process up to (but excluding) timestep t. Such history triplets (i_k, a_k, l_k) consist of the instance i_k, the selected algorithm a_k and some feedback concerning the performance of the algorithm on the instance in terms of an evaluation $l_k = l(i_k, a_k)$, where the loss function l is defined as in the offline AS problem (Equation 2.1).

Depending on how long this process continues, we differentiate between the infinite horizon case, in which no final timestep exists, and the finite horizon case, in which the process ends at some final but unknown timestep T. For simplicity, we exclusively focus on the finite horizon case, where we define the goal as constructing an online algorithm selector s_{online}, which minimizes the *average* loss achieved over the course of time

$$\mathcal{L}_{online}(s_{online}) = \frac{1}{T} \sum_{t=1}^{T} l(i_t, s_{online}(h_t, i_t)) \ . \tag{2.5}$$

The optimal online selector, i.e. *online oracle* or online VBS, can be defined similarly as in the offline AS problem as

$$s^*_{online}(h_t, i_t) = \arg\min_{a \in \mathcal{A}} \mathbb{E}\left[l(i_t, a)\right] \tag{2.6}$$

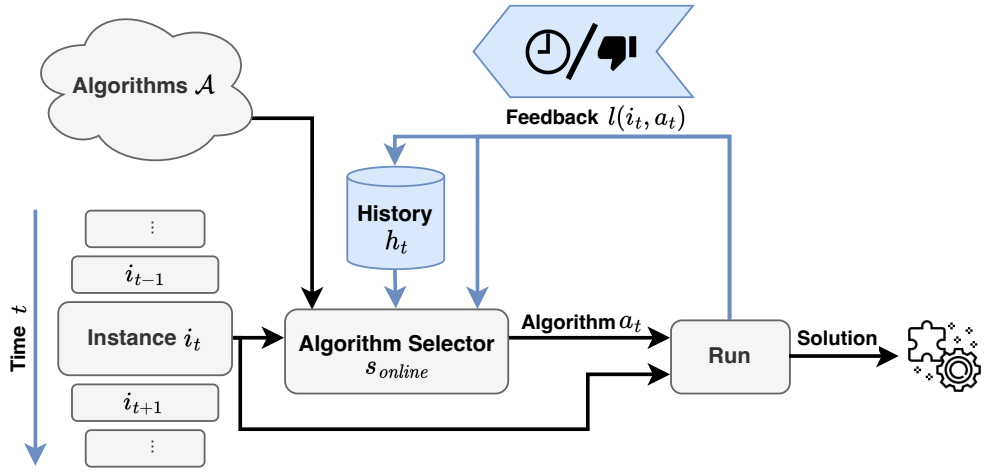

Fig. 2.1: The OAS problem is conducted over several timesteps such that at each timestep an instance $i_t \in \mathcal{I}$ arrives and an algorithm $a_t \in \mathcal{A}$ has to be selected by the online algorithm selector s_{online}. Based on this selection, the selector receives feedback in the form of a loss function evaluation $l(i_t, a_t)$, which is added to the history h_t. The elements making this setting differ from the standard offline AS problem are depicted in blue.

for all timesteps t. Once again, the expectation accounts for any potential randomness included in the algorithm a making $l(i_t, a)$ a random variable.

2.1.3 The Offline Instance-Specific Meta Algorithm Selection Problem

As mentioned in Chapter 1, the AS problem (and its variants) is motivated by the phenomenon of performance complementarity across problem instances, roughly meaning that the best algorithm varies across different instances. Since the AS problem was introduced, several algorithm selectors have been proposed such that the amount of available selectors is large (cf. Section 2.3). As an algorithm selector is an algorithm as well (although with a different task, namely, taking a problem instance as input and returning a presumably well-performing algorithm as output), one may wonder whether selecting among the algorithm selectors themselves can be beneficial. In fact, a certain complementarity across algorithm selectors can indeed be observed (see [Tor+20a]), motivating the meta algorithm selection (meta algorithm selection (MetaAS)) problem, which was first mentioned by Lindauer et al. [LRK19] and Kerschke et al. [Ker+19]. Roughly speaking, the MetaAS problem is concerned with the question: Given a problem instance, a set of algorithms, and a

set of algorithm selectors, which algorithm selector(s) should be used to select the final algorithm for solving the instance?

Of course, such a question could be answered with a selector on the meta level, i.e. an *algorithm selector selector*, which selects an algorithm selector, which in turn selects the algorithm actually solving the problem instance. However, selecting only a *single* algorithm selector is unnecessarily restrictive, as one can also select multiple algorithm selectors and *aggregate* their algorithm choices into a final one in order to (hopefully) make a better final selection. Naturally, this line of thought leads to ensemble learning [Die00], a class of machine learning methods dealing with combining several predictors into a joint, stronger one.

In Chapter 6, we will elaborate on several ways of tackling the MetaAS problem and also demonstrate that the ensemble idea is much more promising than selecting a single selector.

2.1.3.1 Formal Problem Definition

Formally, in addition to all components of the offline AS problem (Section 2.1.1), the MetaAS problem is composed of a set of algorithm selectors $\mathcal{S} \subseteq \{s \mid s : \mathcal{I} \longrightarrow \mathcal{A}\}$. The goal underlying the problem is to find a mapping

$$ass : \mathcal{I} \longrightarrow 2^{\mathcal{S}}\,, \tag{2.7}$$

called *algorithm selector selector*, and an *aggregation function*

$$agg : \mathcal{I} \times 2^{\mathcal{S}} \longrightarrow \mathcal{A}\,, \tag{2.8}$$

such that the algorithm output by aggregation function agg optimizes the loss function l. In other words, we aim at finding the best tuple (agg, ass) of aggregation function agg and algorithm selector selector ass, such that ideally for all instances $i \in \mathcal{I}$ the best algorithm is returned, i.e.

$$agg(i, ass(i)) \in \arg\min_{a \in \mathcal{A}} \mathbb{E}\left[l(i, a)\right] \quad . \tag{2.9}$$

To deepen the understanding of the reader, Figure 2.2 demonstrates the relation between algorithms, algorithm selectors and algorithm selector selectors.

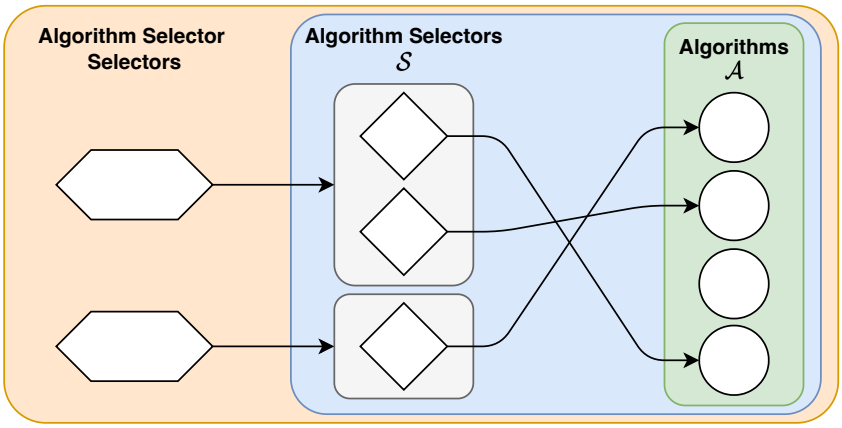

Fig. 2.2: Illustration of the notions algorithms ($\mathcal{A}$), algorithm selectors ($\mathcal{S}$) and algorithm selector selectors. Algorithms solve instances of an algorithmic problem, whereas algorithm selectors select a *single* algorithm from $\mathcal{A}$ given an instance. Finally, algorithm selector selectors select *one or multiple* algorithm selectors, which in turn each select an algorithm. To arrive at a single algorithm to be returned at the end, an aggregation function (not displayed here) aggregates the choices of the different algorithm selectors.

We would like to point out the following aspects related to the definition above:

1. Although the MetaAS problem could, in principle, also be generalized to the online case, i.e. a combination of the MetaAS and the OAS problem, we define it based on the standard (offline) AS problem as all work on the MetaAS problem presented in this thesis is limited to the offline case.

2. In the definition of the aggregation (Equation 2.9), we allow it to depend on the instance, thereby formally opening up the possibility for *learning instance-specific aggregation* functions.

3. Although it might seem counter-intuitive at first sight, we let the aggregation function depend on the powerset of selectors $\mathcal{S}$ instead of the algorithms, as this simplifies the notation in later chapters. For example, defining a weighted aggregation, where each selector receives a certain weight, is easier with the notation above. Nevertheless, the aggregation aggregates the outputs of the different selectors and not the selectors themselves.

2.1.4 Connection Between Problem Variants

We close the presentation of the problem variants by a short summary of the connections between the standard instance-specific AS problem, the OAS problem, and the MetaAS problem in terms of the most distinctive characteristics. Recall that the definition of the corresponding problem always features a distinction from the other problem variants. While the AS problem can be seen as the core and standard problem, OAS is a generalization to the online case in the sense that no upfront training data is available, and correspondingly, the underlying surrogate loss function has to be learned in an iterative fashion online. In contrast to that, MetaAS shifts the problem to the meta level (as the name suggests), no longer directly choosing between algorithms, but rather among selectors, which in turn select an algorithm.

2.2 Distinction From Related Problems

Before discussing how different classes of AS solutions work, let us shortly distinguish the AS problem in its basic form, i.e. the one defined in Section 2.1.1, from other related problems in the domain of automated algorithm design.

Table 2.1 shows a visual representation of some of the differences discussed in the following, in particular focusing on the differences in terms of the prediction target and the algorithmic problem domain.

2.2.1 Algorithm Scheduling

Algorithm scheduling [Hoo+15; Kad+; PT09] can be seen as a generalization of algorithm selection, where, instead of suggesting a single algorithm for an instance, a schedule of algorithms, i.e. a sequence of algorithms mostly in combination with a runtime for each, is suggested. While making this a more complicated problem, it is desirable from the point of view of a practitioner, who in the end wants an instance solved and thus might favor fallback solutions in case a single suggested algorithm cannot solve the instance. It also allows for more flexibility to leverage algorithms that either terminate very quickly or tend to take rather long to solve an instance.

Tab. 2.1: Visual representation of some of the differences between automated algorithm design problems in terms of the prediction target and the algorithmic problem domain.

Autom. alg. design problem	Prediction target					Alg. problem domain	
	Algorithm	Schedule	Configuration	Pipeline	Various	Any	Machine learning
AS	✓					✓	
Algorithm scheduling		✓				✓	
AC			✓			✓	
Meta learning					✓		✓
HPO			✓				✓
AutoML					✓		✓
CASH			✓	✓			✓

Consider an example, visualized in Figure 2.3, with two algorithms a_1, a_2, which always return a valid solution upon termination. Let us assume that a_1 terminates with a probability of 25% at 0.3 seconds, and else it terminates at 20 seconds (with probability 75%). In contrast, a_2 terminates at 2 seconds with a probability of 100%. When choosing an algorithm in the standard AS setting according to its expected runtime, a_2 would be the clear choice as its expected runtime is 2 seconds compared to roughly 15 seconds of a_1. However, in algorithm scheduling, the optimal schedule would be to run a_1 for 0.3 seconds and then switch to a_2 for another 2 seconds as the expected runtime of that schedule would be $0.25 \cdot 0.3 + 0.75 \cdot 2.3 = 1.8$ seconds and thus 0.2 seconds faster than just selecting a_2.

Due to its form, algorithm scheduling solutions often internally compute rankings across the algorithms and then create a schedule such that higher-ranked algorithms come first in the schedule. Algorithm scheduling is often subsumed under the notion of algorithm selection in the literature. However, due to their different prediction targets, we believe a distinction is sensible.

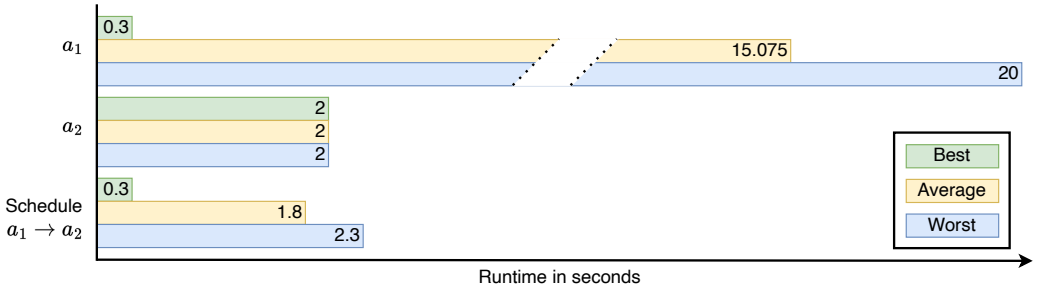

Fig. 2.3: Visualization of an example with two algorithms a_1, a_2, which always return a valid solution upon termination. Algorithm a_1 terminates with a probability of 25% at 0.3 seconds and else it terminates at 20 seconds (with probability 75%). In contrast, a_2 terminates at 2 seconds with a probability of 100%. When choosing an algorithm in the standard AS setting according to its expected runtime, a_2 would be the clear choice as its expected runtime is 2 seconds compared to roughly 15 seconds of a_1. However, in algorithm scheduling, the optimal schedule would be to run a_1 for 0.3 seconds and then switch to a_2 for another 2 seconds as the expected runtime of that schedule would be $0.25 \cdot 0.3 + 0.75 \cdot 2.3 = 1.8$ seconds and thus 0.2 seconds faster than just selecting a_2.

2.2.2 Meta Learning

Meta learning [Van18] is about learning to learn, i.e. learning about the behavior of learning algorithms, often in order to improve them in one way or the other. As such, meta learning is a large field spanning a plethora of topics such as few-shot learning [Wan+20], transfer learning [WKW16], machine learning pipeline composition, and also model selection. AS and meta learning can overlap, if (1) the algorithmic problem is a machine learning problem, (2) the instance space consists of machine learning datasets, and (3) the set of algorithms $\mathcal{A}$ consists of learning algorithms. In such a case, the AS problem is a meta learning problem in the classic sense. However, although the two fields intersect, they also differ from each other as meta learning is much broader in terms of the suggestion targets, whereas AS can also consider other algorithmic problems than machine learning ones.

2.2.3 Algorithm Configuration

Most algorithms expose a set of parameters (also called hyperparameters, if the algorithmic problem is machine learning), which can be adapted to further control the algorithm and fine-tune its behavior. Algorithm configuration (AC) [Sch+22] deals with automatically suggesting suitable parameter configurations for an algorithm for a single (instance-specific AC) or a set of instances (standard AC). Instance-specific

AC can be thought of as a strict generalization of the (instance-specific) AS problem in the sense that the set of algorithms $\mathcal{A}$ is no longer discrete, but potentially infinite and consists of all possible configurations of a single algorithm. Standard AC, in contrast, simplifies this idea by not trying to find a configuration for a particular instance, but rather a configuration, which works well on a (homogeneous) set of instances and, thus, does not need to be adapted. Correspondingly, both instance-specific AS and AC make the assumption that instances from the instance space $\mathcal{I}$ are heterogeneous and, thus, require different algorithms or configurations, whereas standard AC assumes the instances to be rather homogeneous such that a single configuration suffices for all instances.

2.2.4 Hyperparameter Optimization

Hyperparameter optimization (HPO) [Bis+23] can be viewed as a special case of algorithm configuration where the algorithm to configure is a machine learning algorithm. Instead of parameters, one uses the term hyperparameter here in order to distinguish a configurable algorithm parameter from a model parameter, sometimes also called weight. Often, HPO is instance-specific in the sense that a hyperparameter configuration of a machine learning algorithm for a particular problem, i.e. machine learning dataset, should be found.[3]

2.2.5 Automated Machine Learning

The notion of automated machine learning (AutoML) [HKV19] subsumes many aspects related to the vision of automating steps and tasks related to the application of machine learning for a specific task at hand. Arguably, the most studied subproblem of AutoML is the combined algorithm selection and hyperparameter optimization (CASH) problem [Tho+13], where the goal is to compose and configure a concrete machine learning pipeline for a given machine learning dataset with the goal to achieve the best predictive performance possible. Thus, CASH, as the name suggests, is a combination of the HPO and a very specific version of the AS problem for machine learning algorithms. However, research on AutoML covers many more topics such as automated data cleaning and wrangling [VR17; Mah+19], automated

[3] We note that a clear distinction between AC and HPO is not easy and is a frequently discussed topic in the community.

feature engineering [Khu+16] and even automated explanation of models and CASH approaches themselves [Zöl+22; Sas+22; Moo+22]. Although the CASH problem was originally mostly studied on single-label classification tasks, many more tasks can now be solved with CASH tools, for example, multi-label classification [Wev+19; Wev+21] or predictive maintenance [Tor+20b; Tor+21b]. Perhaps, the most prominent subfield of AutoML is neural architecture search (NAS) [EMH19], which deals with finding suitable architectures for neural networks. Depending on the concrete sub-community, the notion of AutoML sometimes also refers to CASH while topics such as automated data cleaning are subsumed by the term of automated data science [Bra+22; Bie+22].

2.3 Common Algorithm Selection Solutions

Since the topic of AS is around for quite some time, naturally, a lot of different approaches to AS have been suggested in the literature. In the following, we elaborate on how all of these approaches are similar in a specific sense and present a framework, which allows one to formalize the majority of existing approaches in a rather simple notation. Note that for the remainder of this section, we focus on the standard offline AS problem (Section 2.1.1) and present existing solutions to that problem. The concepts required to tackle the OAS problem are in principle the same, although the approaches have to deal with different challenges, as we discuss in Chapter 5. Solutions to the MetaAS problem are left out here on purpose as we propose, to the best of our knowledge, the first existing solutions to that problem in Chapter 6.

A good starting point in order to understand how AS approaches work, is the oracle, i.e. the perfect algorithm selector, definition

$$s^*(i) \in \arg\min_{a \in \mathcal{A}} \mathbb{E}\left[l(i,a)\right]$$

originally introduced in Equation 2.3 as it inherently hints at a way of solving the problem despite being a purely theoretical concept. Naturally, we could obtain an empirically very good[4] approximation of the oracle by exhaustively enumerating the set of algorithms and trying each algorithm on a given instance N times in order

[4]Note that it is not necessarily optimal due to a noisy loss function.

to select the one, which performs best on average over the different runs on the corresponding instance, i.e.

$$s(i) \in \arg\min_{a \in \mathcal{A}} \frac{1}{N} \sum_{n=1}^{N} l(i,a) \quad . \tag{2.10}$$

Unfortunately, this is no real solution in practice since (1), as noted earlier, the loss function l is assumed to be costly to evaluate, and (2), even more importantly, enumerating all algorithms defeats the purpose of the algorithm selection altogether — let alone executing the same algorithm multiple times. In the end, we are interested in the solution for the instance produced by the best algorithm, which we would already obtain during the process of choosing the algorithm. Correspondingly, by the time we have solved the AS problem in the form of the corresponding instance, the instance itself would already be solved by the best algorithm available to us. However, this instance solution comes at extremely large costs as all algorithms have to be run multiple times in the strategy above. In the case of constraint satisfaction problems, such as SAT, where the most commonly considered loss function is the time until solution, the strategy above would always yield a worse result than choosing any algorithm at random.

Note that the problems of the naive solution approach above arise from the fact that l is costly to evaluate, or in other words that, in order to evaluate l for a particular algorithm a, the algorithm has to be run. If the loss function l was very cheap to evaluate, we could simply enumerate all algorithms as the overhead would be negligible. Consequently, the solution above can be made viable by replacing l with a surrogate $\widehat{l} : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}$ featuring the same signature, but being cheap to evaluate. Assuming that $\widehat{l}$ is a deterministic function, we can even dispense with the multiple evaluations shown in Equation 2.10 yielding a canonical algorithm selection strategy defined as

$$s(i) \in \arg\min_{a \in \mathcal{A}} \widehat{l}(i,a) \quad . \tag{2.11}$$

Naturally, this selector definition via a surrogate raises two important questions:

1. What should $\widehat{l}$ look like, i.e. which properties should it fulfill?

2. How to obtain $\widehat{l}$?

In the following, we answer these questions in a systematic manner.

2.3.1 Desired Properties of Surrogate Loss Functions

Before we delve deeper into the large variety of existing methods on learning surrogate functions, i.e. answer Question 2, we will first introduce desirable properties such surrogate functions should have, i.e. answer Question 1.

First, as already mentioned, a good surrogate loss function in our context should be cheap to evaluate, where the exact definition of cheap varies depending on the algorithmic problem tackled and the associated true loss function l. Consider, once again, the problem of SAT, where one often tries to find the algorithm solving a given instance the fastest. In such a case, $\widehat{l}$ should be fast to evaluate in terms of time.

Second, ideally, a surrogate loss function should be order-preserving such that if the original loss function imposes a certain ordering on the algorithms for a given instance, the surrogate loss function should impose the same ordering. Formally, we want that for all instances $i \in \mathcal{I}$ and any two algorithms $a_1, a_2 \in \mathcal{A}$ that it holds that

$$l(i, a_1) \leqslant l(i, a_2) \Rightarrow \widehat{l}(i, a_1) \leqslant \widehat{l}(i, a_2) \ . \tag{2.12}$$

In principle, this property can even be weakened such that it does not have to hold for any two algorithms, but rather for the best algorithm

$$a_i^* \in \arg\min_{a \in \mathcal{A}} l(i, a)$$

and any algorithms $a \in \mathcal{A}$:

$$l(i, a_i^*) \leqslant l(i, a) \Rightarrow \widehat{l}(i, a_i^*) \leqslant \widehat{l}(i, a) \ . \tag{2.13}$$

In other words, the weakened property requires that for each instance the best algorithm according to the original loss function l is also the best algorithm according to the surrogate loss function $\widehat{l}$. The stronger version of this property is of interest, if the algorithm selector is not only asked to predict a single algorithm, but potentially a ranking across the algorithms (see, for example, Chapter 3). Many algorithm selectors do this internally anyway due to the surrogate loss function being used.

The interested reader might have noticed that the canonical AS strategy (Equation 2.11) equipped with a surrogate function fulfilling Properties 1 and 2 (weak or strong version) directly yields the oracle selector s^*. Unfortunately, Property 2

is often violated at least for some part of the instance space $\mathcal{I}$ in practice for all surrogate functions such that the canonical algorithm selection strategy only approximates the oracle. Naturally, this violation arises as the surrogate loss function in practice only approximates the actual loss functions due to a variety of practical problems such as noise in the training data, lack of training data, or insufficiently informative instance features. Similarly, choosing an inappropriate model class for the surrogate loss function can lead to a difference between the surrogate and the actual loss function.

2.3.2 Learning Surrogate Loss Functions

The answer to the second question, i.e. how to obtain $\widehat{l}$, is easy: Machine learning. Of course, we could in principle ask a domain expert to define rules when to apply which algorithm and define a surrogate loss function based on these rules. However, we, and so does the literature, focus on algorithm selection approaches learned from data. This is done as even domain experts often have trouble knowing upfront, which algorithm will perform best for a given instance. Moreover, even if such experts exist, they are, at best, sparsely available.

Although often not the case in practice, note that for the remainder of this section we assume for simplicity that we have evaluations $l(i, \cdot)$ for all algorithms on all training instances $i \in \mathcal{I}_D$. Chapter 3 and Chapter 4 will touch on how to handle the problem of missing evaluations (due to different causes) in more detail.

We start by defining the arguably most simple data-driven algorithm selector, called the single best solver (SBS). The SBS is a feature-free approach in the sense that it does not require any instance features computed by f since its prediction is not instance-specific. It simply always suggests the algorithm, which performed best on the training instances $\mathcal{I}_D$, i.e.

$$s_{SBS}(i) \in \arg\min_{a \in \mathcal{A}} \widehat{l}_{SBS}(i, a) \tag{2.14}$$

with

$$\widehat{l}_{SBS}(i, a) = \frac{1}{|\mathcal{I}_D|} \sum_{i' \in \mathcal{I}_D} l(i', a) \ . \tag{2.15}$$

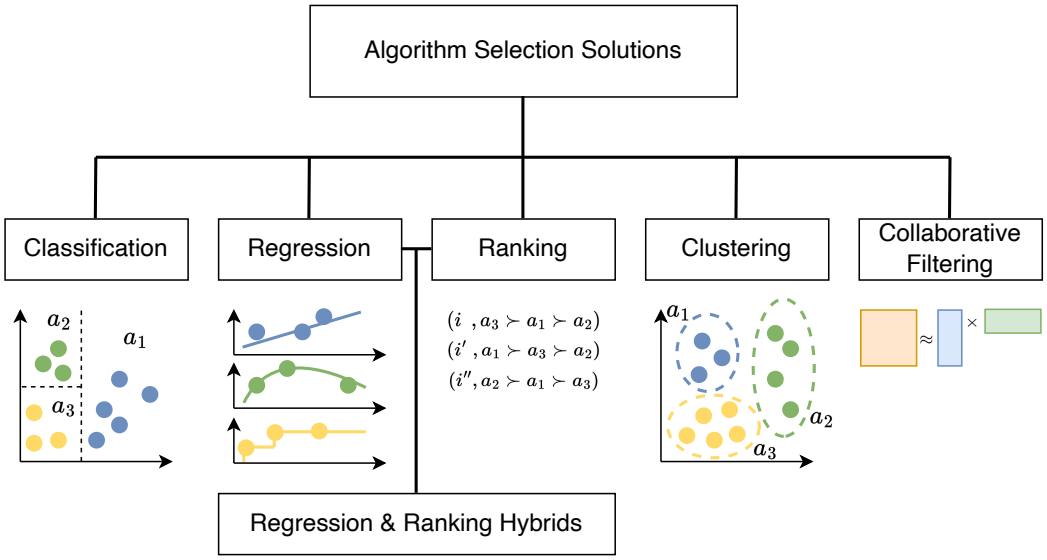

Fig. 2.4: Taxonomy of different algorithm selection solutions.

Due to its simplicity, the SBS is often considered as a baseline for algorithm selectors in the sense that any reasonable algorithm selector should at least perform better than the SBS.

In the following, we will present a taxonomy of different algorithm selection solutions based on the type of learning problem tackled to learn the surrogate loss function and give some examples of algorithm selectors leveraging loss functions of the corresponding kind. Thereby, we mainly focus on the machine learning component of the corresponding approach. Furthermore, we do not claim the list of approach categories and approaches to be exhaustive, but instead focus on formalizing several common approach classes under a unified framework and then give a small set of examples for each of such categories. A visual overview of the taxonomy is presented in Figure 2.4. Although we believe that the following sections are more than enough to get a deep understanding of AS approaches, we refer to Kotthoff [Kot16] and Kerschke et al. [Ker+19] for a complete overview of existing AS literature.

2.3.2.1 Classification Solutions

Classification-based algorithm selection approaches work by directly trying to learn the selector $s : \mathcal{I} \longrightarrow \mathcal{A}$ through the means of classification instead of learning a surrogate loss function. Correspondingly, assuming that $|\mathcal{A}| > 2$, they solve a

multi-class classification problem. Although they do not explicitly learn a surrogate loss function, they can be defined in a way that fits the framework introduced above as follows:

$$s_{classification}(i) \in \arg\min_{a \in \mathcal{A}} \widehat{l}_{classification}(i, a) \tag{2.16}$$

where

$$\widehat{l}_{classification}(i, a) = \begin{cases} 0 & \text{if } h(\boldsymbol{f}_i) = a \\ 1 & \text{else} \end{cases} \tag{2.17}$$

Here, $h : \mathbb{R}^d \longrightarrow \mathcal{A}$ is the actual classification model predicting a suitable algorithm based on an instance feature vector $\boldsymbol{f}_i$. The training dataset $\mathcal{D}_{classification}$ required to learn the classifier h can be constructed from the training instances (for which evaluations are available), as

$$\mathcal{D}_{classification} := \{(\boldsymbol{f}_i, a^*) \mid i \in \mathcal{I}_D \wedge \forall a \in \mathcal{A} : l(i, a^*) \leqslant l(i, a)\} \subset f(\mathcal{I}) \times \mathcal{A} \tag{2.18}$$

With such a dataset, essentially any (multi-class) classification technique can be employed to solve the problem. This approach is denoted as MultiClass or MultiClassSelector throughout the experiments in this thesis.

There exist various approaches in the literature, which directly follow this scheme using different kinds of machine learning models to model h such as decision trees (e.g., [Gen+10; GM04]), k-nearest neighbor models (e.g., [Mal+11]) or support vector machines (e.g., [HW06]). Arguably, the most famous example from this category is SATzilla'11 [Xu+11], which employs an all-pairs (one-vs-one) decomposition approach [LCG08] to the multi-class classification problem, learning a cost-sensitive classifier for each pair of algorithms and determining the selected algorithm by majority voting[5]. Moreover, some approaches employ the classification idea on top of other loss function surrogates as done by Kotthoff [Kot12] — an idea, which we further explore in Chapter 6.

In practice, this direct approach of learning an algorithm selector tends to be subpar in the sense that it yields rather bad algorithm selections as one can also see in the evaluations of Section 4.1 and Chapter 6. However, SATzilla'11 is a good counterexample as it is still among the state-of-the-art AS tools. A clear reason why classification approaches often fail in AS is — to the best of our knowledge — unknown. However, considering the success of SATzilla'11 with its cost-sensitivity and regression-based approaches (cf. Section 2.3.2.2), we hypothesize that the

[5] We note that one could also classify SATzilla'11 as a ranking approach leveraging pairwise comparisons (cf. Section 2.3.2.3).

reason for this is grounded in the loss of information induced by the binary or categorical labels in the training data compared to the true loss values. It seems that the information contained in the concrete loss value associated with an algorithm on an instance makes a difference (cf. Equation 2.18 and Equation 2.19).

2.3.2.2 Regression Solutions

Instead of treating the AS problem as a classification problem, the canonical AS strategy (Equation 2.11) suggests treating it as a multi-target regression problem with one regression target for each algorithm. This regression target is the algorithm's loss function value according to l. Correspondingly, we can also aim to learn a multi-target regression function

$$h_{regression} : \mathbb{R}^d \longrightarrow \mathbb{R}^{|\mathcal{A}|}$$

where we assume for ease of notation that $h_{mr}(\boldsymbol{f}_i)_a$ gives us the predicted loss function value for algorithm $a \in \mathcal{A}$ on instance i. The dataset required to learn such a function can be constructed from the training instances (and available evaluations on them) as

$$\mathcal{D}_{regression} := \left\{ \left(\boldsymbol{f}_i, \left[l\left(i, a_1\right), \ldots, l\left(i, a_{|\mathcal{A}|}\right) \right] \right) \mid i \in \mathcal{I}_D \right\} \subset f(\mathcal{I}) \times \mathbb{R}^{|\mathcal{A}|} \tag{2.19}$$

With such a function, we can then define the algorithm selector and the corresponding surrogate loss function as

$$s_{regression}(i) \in \arg\min_{a \in \mathcal{A}} \widehat{l}_{regression}(i, a) \tag{2.20}$$

and

$$\widehat{l}_{regression}(i, a) = h_{regression}(\boldsymbol{f}_i)_a \ . \tag{2.21}$$

With this definition, the underlying multi-target regression problem can be used with any method amendable to that problem. In the practice of AS a commonly used (quite strong) baseline, called the *PerAlgorithmRegressor* in this thesis, decomposes the multi-target regression problem into separate standard regression problems, learning one regressor for each algorithm independently of the remaining ones as depicted in Figure 2.5.

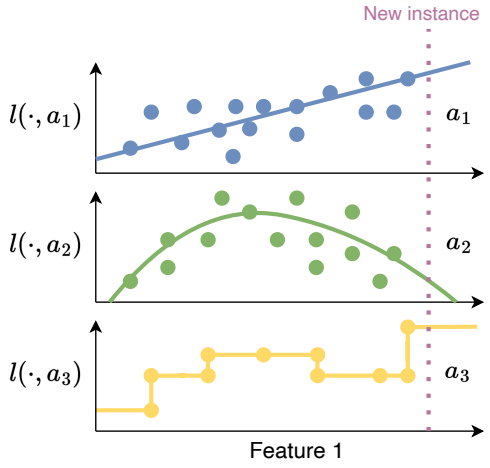

Fig. 2.5: Decomposition of multi-target regression AS problem formulation, where one regressor is learned separately for each algorithm. We assume that instances are represented only by a single feature. The training data points are depicted in different colors corresponding to different algorithms, the learned models are represented by lines. The feature of a new instance i_{new}, for which a selection should be made, is depicted as a pink dashed vertical line. Here, a_2 would be chosen according to the learned regressors as $h_{mr}(\boldsymbol{f}_{i_{new}})_{a_2}$ yields the lowest loss value.

One of the most famous examples of AS approaches following this technique is called SATzilla, which in its first two versions SATzilla'04 [Nud+04a] and SATzilla'07 [Xu+08] — both predecessors of SATzilla'11 — employs one regression model for each algorithm for predicting its corresponding loss function value. In general, a large set of literature exists, which suggests methods leveraging regression models to predict the performance of an algorithm on an instance [HW09; RHF07; Hut+06; LNS02]. The problem of runtime prediction in particular received a significant amount of attention during the last decade (see [Hut+14] for an overview).

Naturally, while being simple and yielding a surprisingly good AS strategy in practice, the algorithm-wise decomposition also carries the disadvantage that it ignores dependencies/correlations between algorithms and their loss function values. Considering that many algorithms are based upon each other (consider for example the SAT domain where the most successful algorithms are variations of the same base algorithm), this approach can be disadvantageous, especially in cases where training data is limited (cf. Chapter 3). Moreover, posing the problem as a regression problem aims at trying to learn a surrogate loss function, which predicts the true loss function values as correctly as possible. However, recalling the second desired property of surrogate loss functions, trying to mimic the exact loss function as closely as possible is a sufficient criterion for having order preservation, but not a necessary

one. Correspondingly, one may wonder whether by solving a regression problem of the kind above we solve a problem more difficult than actually necessary. This line of thought naturally leads to the idea of posing the problem as a ranking problem, which we discuss next.

2.3.2.3 Ranking Solutions

As mentioned earlier, ranking-based AS solutions are motivated by the observation that the ability to predict a correct ranking across the algorithms for a given instance is potentially an ability easier to achieve than the ability to correctly predict the true loss function value for each algorithm. Here, the goal is to learn a function

$$h_{ranking} : \mathbb{R}^d \longrightarrow \mathcal{R}(\mathcal{A}) \ , \tag{2.22}$$

that, given instance features, predicts a ranking across the algorithms such that the algorithm ranked at position one is the best one and the one ranked at the last position is the worst one, where $\mathcal{R}(\mathcal{A})$ is the set of possible rankings over the set of algorithms $\mathcal{A}$. Correspondingly, the problem we are trying to solve here is a label ranking problem [VG10], where each algorithm corresponds to a label and the context is defined by the instance (features). The function can be learned using a dataset of the form

$$\mathcal{D}_{ranking} := \left\{ \left(\boldsymbol{f}_i, [a_{\pi_i(1)} \succ \ldots \succ a_{\pi_i(|\mathcal{A}|)}) \right] \mid i \in \mathcal{I}_D \right\} \subset f(\mathcal{I}) \times \mathcal{R}(\mathcal{A}) \tag{2.23}$$

where $a_1 \succ a_2$ means that a_1 is preferred over a_2 and $\pi_i : \mathbb{N} \longrightarrow \mathbb{N}$ gives the ID of the algorithm, which is ranked at the given position for instance i according to the loss function l. Note that we assume here, for simplicity of notation, that the loss function value of no two algorithms is the same for an instance, which in practice is often not the case. Nevertheless, one can adjust the definition of the dataset above such that ties between algorithms are allowed.

With such a function, we can then define the algorithm selector and the corresponding surrogate loss function as

$$s_{ranking}(i) \in \arg\min_{a \in \mathcal{A}} \widehat{l}_{ranking}(i, a) \tag{2.24}$$

and

$$\widehat{l}_{ranking}(i, a) = h_{ranking}(\boldsymbol{f}_i)_a \ ,$$

where we assume that $h_{ranking}(\boldsymbol{f}_i)_a$ gives the position of algorithm a in the predicted ranking $h_{ranking}(\boldsymbol{f}_i)$.

Once again, the problem of learning the label ranking function can, in principle, be tackled by any label ranking algorithm that can potentially handle ties between labels, i.e. algorithms. If the ranking model is based on a latent utility function, as in the case of the Plackett-Luce model [CHD10], the most straightforward way to tackle the problem above is a decomposition enabling to learn one latent utility function per algorithm similar to the regression decomposition discussed earlier.

In practice, existing methods apply different label ranking approaches to the problem. For example, Cunha et al. [CSC18] evaluate a set of label ranking approaches based on nearest-neighbor strategies or label ranking trees [CHH09]. Similarly, Kotthoff [Kot14] empirically evaluate different approaches to convert the predictions of machine learning models into rankings and thus also inherently solve a label ranking problem. As a last example, Oentaryo et al. [OHL15] employ a probabilistic ranking model trained based on a likelihood formulation of the rankings available in the training data. Although most AS approaches are inherently able to generate a ranking over the complete set of algorithms (e.g. regression-based approaches), the majority of existing work does not evaluate the capability to do so in the classical AS problem as defined in Section 2.1.1. A larger amount of explicit ranking strategies can be found in the literature considering the algorithm scheduling problem (cf. Section 2.2.1), which, depending on the concrete problem form, can be cast as a ranking problem.

One of the main downsides of ranking based AS approaches is that they often ignore a large part of the numerical loss function values available for the training instances as they only take into account the ranking across algorithms based on these values instead of the values themselves for training. This leads to a loss of information, which could be used as part of the training process. Consider, for example, a case where two algorithms, say a_1 and a_2, feature very similar loss function values on an instance i, say $l(i, a_1) = 2.33$ and $l(i, a_2) = 2.34$. While a regression-based approach would at least in principle be able to capture that these two algorithms perform very similarly on the instance and in practice there might not be much of a difference at all, the training dataset (Equation 2.23) provided for a ranking approach is unable to capture this information as we would only have that $a_1 \succ a_2$. Although more advanced label ranking algorithms can in principle handle ties, one would need to define how close two algorithms are allowed to be wrt. their loss function values in

order to justify a tie in the training data, which introduces another hyperparameter that could largely influence the success of an algorithm selector.

2.3.2.4 Hybrids of Ranking and Regression Solutions

Quickly revisiting the regression and ranking-based solutions, one could argue that the two are extremes of the same idea of being able to rank algorithms. While the regression idea takes the one extreme by trying to mimic the actual loss function as closely as possible and thus inherently generates a ranking, the ranking idea takes the other extreme in the sense that it just wants to achieve the correct ranking, but has no intention and often also no possibility of correctly predicting the actual loss function values. Correspondingly, one may wonder whether hybrids of ranking and regression approaches might be a good idea to alleviate each other's weaknesses — a concept, which has been demonstrated effective on different tasks (not including AS) by Sculley [Scu10].

Following the strategy of many ranking approaches, the general idea here is to learn a utility function

$$u : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R} \tag{2.25}$$

on the basis of which a ranking can be constructed at the end. However, in contrast to ranking approaches, the utility function is learned based on a combination of a ranking loss function $\mathcal{L}_{ranking}$, such as the hinge ranking loss [SS06], and a regression loss function $\mathcal{L}_{regression}$, such as the squared error. Leveraging such a utility function u, we can then define both the corresponding selector and the surrogate loss function as

$$s_{hybrid}(i) \in \arg\min_{a \in \mathcal{A}} \widehat{l}_{hybrid}(i, a) \tag{2.26}$$

and

$$\widehat{l}_{hybrid}(i, a) = -u(\boldsymbol{f}_i, a) \ .$$

This idea was proposed by Hanselle et al. [Han+20] who suggest learning one utility function separately per algorithm (similar to the decomposition approach presented for the regression models), based on a convex combination of regression and ranking loss functions. It was later refined by Fehring et al. [FHT22] who learn

such a function across algorithms[6]. According to the authors, this can indeed yield better algorithm selection performance in some cases.

2.3.2.5 Clustering Solutions

Clustering-based AS approaches differ considerably from the previously seen approach classes as they can be seen as decomposition approaches. They decompose the algorithm selection on the complete instance space to multiple AS problems for different (possibly overlapping) subregions of the instance space $\mathcal{I}$ or more precisely the feature space $f(\mathcal{I}) \subseteq \mathbb{R}^d$ as Figure 2.6 depicts. To achieve this decomposition, clustering-based AS approaches firstly cluster the training instances based on a distance function such that at the end K different clusters of instances exist. The concrete distance function, the form of clusters and the number of clusters are either hyperparameters or inherently defined by the clustering algorithm used and differ depending on the concrete approach as we will shortly see. After these clusters are obtained, clustering-based approaches then can train one of the previously described AS approaches for each cluster. Then, when the approach is asked to select an algorithm for a new instance, the closest cluster according to the distance function is determined and the local AS model trained on the corresponding cluster is queried to select the algorithm.

For ease of notation let us assume that we have a function

$$h_{cluster} : \mathbb{R}^d \longrightarrow \mathcal{C} \tag{2.27}$$

mapping from the instance feature space to the set of clusters $\mathcal{C}$ previously learned, which, given an instance in the form of its feature vector to its corresponding closest cluster. Then, if we assume that

$$\widehat{l}_{local} : \mathcal{I} \times \mathcal{A} \times \mathcal{C} \longrightarrow \mathbb{R} \tag{2.28}$$

returns the value of the *local* loss function surrogate learned for a given cluster $c \in \mathcal{C}$, the surrogate loss function and the corresponding algorithm selector can be defined as

$$s_{cluster}(i) \in \arg\min_{a \in \mathcal{A}} \widehat{l}_{cluster}(i, a) \tag{2.29}$$

[6]Although the author of this thesis is also an author of the corresponding papers, we do not list these as contributions of this thesis as the author of this thesis is not the main author of the papers.

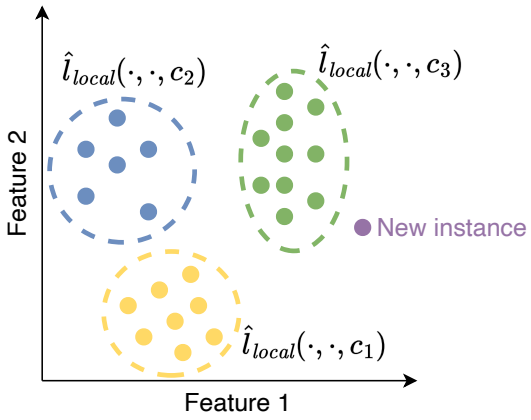

Fig. 2.6: General idea of clustering-based AS approaches where we assume instances to be represented by two features. Training instances are clustered according to their features by some clustering algorithm. Then, for each cluster, a local surrogate loss function of any kind is learned. If a new instance (depicted in magenta) arrives, the closest cluster (c_3 in this case) is determined and the presumably best algorithm according to the local loss function surrogate associated with that cluster is selected.

and

$$\widehat{l}_{cluster}(i,a) = \widehat{l}_{local}(i,a,h_{cluster}(\boldsymbol{f}_i)) \ .$$

Note that the local loss function surrogates for each cluster can in principle be of any of the previously discussed functions and thus can be instantiated by any previously discussed class of AS approaches.

Examples following this strategy are ISAC [Kad+10], which leverages g-means clustering [HE03] and is originally designed to perform instance-specific algorithm configuration. However, it can be easily adapted as an algorithm selection strategy by returning the algorithm performing best on average on the instances of the cluster closest to a new given instance. Similarly, Malitsky et al. [Mal+13] propose CSHC (originally for instance-specific algorithm configuration), a cost-sensitive hierarchical clustering technique, which determines clusters based on the consensus about the best algorithm among instances and then suggests the corresponding consensus algorithm of the closest cluster when given a new instance. As a last example, but without any upfront clustering, SUNNY [AGM14] collects the closest instances leveraging a k-nearest neighbor approach given a new instance and then returns the algorithm performing best on average on these neighboring instances.

As clustering based AS systems can in principle use different AS approaches for the different clusters (although this is not done in practice), they are very versatile and highly configurable and thus also highly adaptable. Unfortunately, this large amount of hyperparameters can also be seen as a negative point since they have to be set correctly in order to achieve a good performance. Moreover, due to the rather local AS models being learned for each cluster, dealing with instances, which are far away from these clusters (outliers) can be problematic. For this reason, some approaches define a fallback strategy for such instances, which can make the approach deteriorate in practice as we explain in more detail in Section 6.4. Lastly, likewise nearest neighbor and clustering techniques in general, these kinds of approaches are very susceptible to non-informative features as they have no means of down-weighing a feature, if it carries a low amount of information.

2.3.2.6 Collaborative Filtering Solutions

Collaborative filtering (CF) [SK09] based AS solutions are motivated by treating the AS problem as a recommendation problem similar to, for example, recommending products (i.e. algorithms in our case) to customers (i.e. instances in our case). A fundamental concept is the so-called (usually sparse) rating or performance matrix $R^{|\mathcal{I}_D|\times|\mathcal{A}|}$, where an entry $R_{i,a} = l(i,a)$ corresponds to the performance of algorithm a on instance $i \in \mathcal{I}_D$ according to l if known, and $R_{i,a} = ?$ otherwise. A visualization can be seen in Figure 2.7. Common problems to tackle in the field of recommender systems are then either predicting missing entries in this matrix (e.g. if a customer likes a certain product or not) or, known as the cold-start problem, predicting a complete row in case a new customer (i.e. instance) is added to the matrix. Obviously, the second problem is of more interest in the case of AS, where we are interested in choosing the best algorithm for a previously unseen instance. In general, CF approaches can be categorized into two kinds of approaches: memory-based or model-based.

Memory-based approaches work by quantifying similarities between instances by applying a similarity function (or equivalently a distance function) either to the instance features or directly to the loss function evaluations available for the instances. Then, when a new instance arrives, the loss function value for each algorithm is predicted as the aggregation of the values associated with a set of similar instances, which are available in the performance matrix R. More formally expressed in the

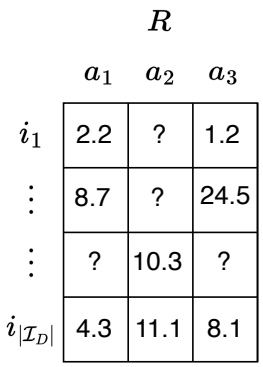

Fig. 2.7: Example of a rating matrix with one row per training instance and a column per algorithm. Each cell $R_{i,a}$ contains the loss of algorithm a on instance i according to a given loss function l. Missing values are depicted by a '?'.

notation established so far, the surrogate loss function value for any algorithm $a \in \mathcal{A}$ for a new instance $i \in \mathcal{I}$ can be computed as

$$\widehat{l}_{CF-memory}(i, a) = agg\left(\left\{R_{i',a} \mid i' \in \Gamma(i)\right\}\right) \tag{2.30}$$

such that

$$s_{CF-memory}(i) \in \arg\min_{a \in \mathcal{A}} \widehat{l}_{CF-memory}(i, a) \tag{2.31}$$

is the corresponding algorithm selector.

Here, $\Gamma(i) : \mathcal{I} \longrightarrow 2^{\mathcal{I}}$ is a neighborhood function returning a set of instances, which are similar to the given instance according to some defined similarity function and $agg : 2^{\mathbb{R}} \longrightarrow \mathbb{R}$ aggregates a set of loss function values into a single one. Popular choices for the similarity functions include measures quantifying correlations between variables such as the Pearson correlation [CB21] or standard vector similarity functions such as the cosine-similarity [SK09]. Aggregation can be performed with virtually any aggregation function such as the average or sum.

As mentioned, the neighborhood function Γ can either work based on performance matrix entries or based on instance features. In our case, i.e. the cold-start problem, there are no performance entries available for a new instance such that similar instances have to be computed based on the instance features. Correspondingly, memory-based CF approaches are very similar to clustering-based approaches discussed in the previous section and to any approach working with nearest-neighbor-based algorithms. As a consequence, they suffer from similar weaknesses. In particular, the predictions are rather based on local trends and thus are in danger of missing subtle global trends as Koren [Kor08] argues. However, this can also be

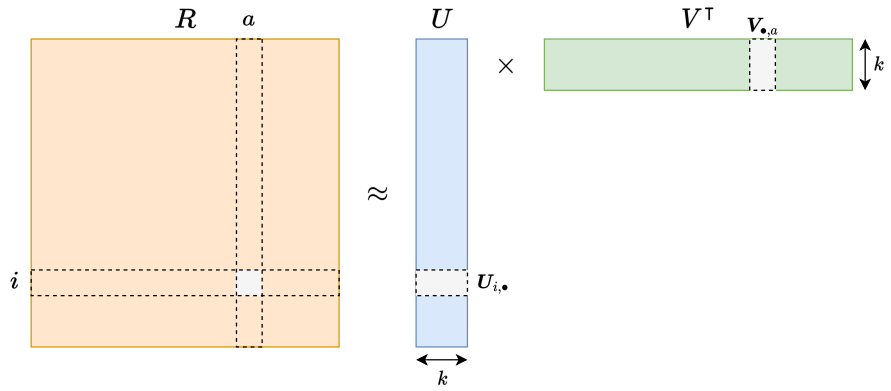

Fig. 2.8: Visualization of the matrix decomposition for model-based collaborative filtering. The performance matrix R is decomposed into two matrices $U \in \mathbb{R}^{|\mathcal{I}_D| \times k}$ and $V \in \mathbb{R}^{k \times |\mathcal{A}|}$. The latent features of an instance i are given by $\boldsymbol{U}_{i,\bullet} \in \mathbb{R}^k$ and by $\boldsymbol{V}_{\bullet,a} \in \mathbb{R}^k$ for algorithm a.

seen as advantageous since the predictions are highly specialized and not disturbed by potentially irrelevant information associated with dissimilar instances.

In contrast, model-based CF approaches work by learning a model on the performance matrix R as the name suggests. While there exists a plethora of approaches to CF in general [SK09], we focus on matrix decomposition techniques, which have been used in the context of AS. They work by decomposing the performance matrix R into two matrices $U \in \mathbb{R}^{|\mathcal{I}_D| \times k}$ and $V \in \mathbb{R}^{k \times |\mathcal{A}|}$ w.r.t. some loss function $L(R, U, V)$, such that

$$R \approx \hat{R} = UV^\top , \tag{2.32}$$

where U can be interpreted as latent features of the instances and V as latent features of the algorithms, and k is the number of latent features. In principle, any matrix decomposition technique such as a singular value decomposition (SVD) can be used to solve the decomposition problem. Often, a corresponding technique comes with an associated loss function $L(R, U, V)$. However, one should consider that techniques dealing with missing values[7] are preferable since the performance matrix is often sparsely filled. If one works with techniques, which cannot handle missing values, memory-based CF approaches can be applied before decomposing the matrix in order to fill in missing values. Obviously, this bears the danger of error propagation. A visualization of the decomposition idea can be seen in Figure 2.8.

[7] Note that SVD in its standard form cannot deal with missing values.

Having latent features $\boldsymbol{U}_{i,\bullet} \in \mathbb{R}^k$ for an instance i and $\boldsymbol{V}_{\bullet,a} \in \mathbb{R}^k$ for each algorithm $a \in \mathcal{A}$, we can predict the loss value as

$$\widehat{l}_{CF-model}(i,a) = (\boldsymbol{U}_{i,\bullet})^\intercal \boldsymbol{V}_{\bullet,a} \tag{2.33}$$

yielding an algorithm selector as

$$s_{CF-model}(i) \in \arg\min_{a \in \mathcal{A}} \widehat{l}_{CF-model}(i,a) \ . \tag{2.34}$$

In principle, instead of using the linear model based on the latent features shown in Equation 2.33, one can even learn any machine learning model on top (as also suggested by Malitsky and O'Sullivan [MO14]), i.e. perform a form of stacking [Wol92], to potentially obtain more precise estimates of the underlying target loss function l.

Unfortunately, these latent instance features are computed once when decomposing the performance matrix spanned by the training instances and are thus unavailable for new instances making the computation of $\widehat{l}_{CF-model}$ as defined above infeasible. One solution to this problem is to simply recompute the complete decomposition every time a new instance arrives and attach this instance to the matrix. However, this is not only computationally expensive (and thus yields very long algorithm selection times), but also bears the danger of yielding bad loss function surrogate estimates as the complete performance row associated with the instance is empty before the decomposition. A second solution was suggested by Malitsky and O'Sullivan [MO14] and later also Misir and Sebag [MS17], who propose to solve the cold-start problem by learning a function $h_{instance} : \mathbb{R}^d \longrightarrow \mathbb{R}^k$ mapping from the original instance feature space to the latent instance feature space such that the latent features $\boldsymbol{U}_{i,\bullet} = h_{instance}(\boldsymbol{f}_i)$ of an instance can be predicted upon arrival based on the computed instance features $\boldsymbol{f}_i$.

The first work formalizing the AS problem as a CF problem is by Stern et al. [Ste+10] who suggest a model-based approach rooted in a probabilistic loss function quantifying the quality of the performance matrix decomposition. Later, Misir and Sebag [MS13; MS17] suggest a memory-based and a model-based CF AS approach. The memory-based approach measures the similarity between problem instances using the cosine similarity of the associated performance matrix rows whereas the suggested model-based approach leverages CoFi$^{\text{RANK}}$ [Wei+07] to decompose

the performance matrix optimizing the ranking across algorithms via the NDCG [Wan+13] ranking loss function.

While memory-based CF approaches mostly capture local trends, model-based CF approaches are better at capturing global trends [Kor08]. Solving the cold-start problem is much harder for model-based approaches than for memory-based ones as the only really suitable solution in the context of AS is the one originally suggested by Malitsky and O'Sullivan [MO14] (and later used by Misir and Sebag [MS13; MS17]) in the form of learning a mapping from the instance feature space to the latent feature space, which is a rather difficult problem to solve upfront. This is particularly unfortunate, as Malitsky and O'Sullivan [MO14] have shown that instantiating the surrogate loss function by a linear model trained based on the ground truth latent features can in practice reach oracle performance. Obviously, this assessment is an optimistic one as the ground truth latent features can only be obtained from the complete performance matrix, meaning that they are based (in parts) on the actual test data. However, this is unavoidable, if one wants to quantify the quality of the ground truth latent features in terms of their predictive power in the context of AS. When trying to predict the latent features based on the actual instance features as suggested, Malitsky and O'Sullivan [MO14] observe that the performance is much worse and similar to standard regression-based AS approaches.

2.3.2.7 Further Automating Algorithm Selection

As the previous sections have shown, there exists a large variety of different AS approaches, which learn a surrogate loss function based on very different kinds of strategies. Furthermore, most of these strategies have a large variety of hyperparameters and components, which in turn can be implemented using a plethora of sub-approaches. Naturally, this leads to a set of questions such as when to apply which AS approach and how to correctly configure it. While we try to answer the first question in Chapter 6 on an instance level, one can also pose this question on the level of scenarios, i.e. sets of instances. This problem of both selecting and configuring the AS approach for a given set of instances is tackled by AutoFolio [Lin+15], which is extremely effective. Similar problems have been tackled in related problems such as AutoML (cf. Section 2.2.5) [FH18].

2.4 Algorithm Selection Loss Functions for Common Algorithmic Problem Classes

When considering loss functions for the AS problem, it is important to distinguish between two kinds of algorithmic problems for which an algorithm should be selected. On the one hand, there are so-called constraint satisfaction problems for which *any* solution adhering to a set of given constraints should be found. On the other hand, there are so-called (constrained) optimization problems, for which a solution should be found, which (potentially) also adheres to a set of constraints, but in addition is as good as possible according to a given objective function. This distinction is important as one often considers different loss functions in the context of AS for these problem types, calling for different AS approaches. In the following, we discuss how the two problem classes differ from the perspective of AS and correspondingly, why often different loss functions are used. As part of this discussion, we provide an overview of common loss functions.

Once again, we focus on the offline instance-specific AS problem (cf. Section 2.1.1) for this section for ease of understanding. Most of the presented loss functions are easily transferable to the online setting as they are naturally instance-wise decomposable.

When describing losses formally in the following, we will denote them as functions following the signature

$$\mathcal{L} : 2^{\mathcal{I}} \times \mathcal{S} \longrightarrow \mathbb{R} \ . \tag{2.35}$$

Hence, they return the loss of a specific selector s on a finite set of instances $I \subset \mathcal{I}$, also called scenario.

2.4.1 Algorithm Selection Loss Functions for Constraint Satisfaction Problems

Formally, a constraint satisfaction problem can be defined as a triple (Z, R, B) where $Z = \{Z_1, \ldots, Z_n\}$ is a set of variables, $R = \{R_1, \ldots, R_n\}$ is a set of domains, R_i is the domain of variable Z_i, and $B = \{B_1, \ldots, B_m\}$ is a set of constraints essentially defining which variables can be assigned what values from their corresponding

domain. The solution to such a problem is an assignment of values to all variables in Z such that all constraints in B are fulfilled.

As the name of the problem suggests, solutions either satisfy the constraints and thus are feasible or they do not satisfy the constraints and thus are infeasible. Consequently, constraint satisfaction problems come with a binary notion of suitability or quality of a solution. Under the assumption that an algorithm only returns feasible solutions, the most important quantity to optimize for is the time until such a solution is found or the number of solved instances within a certain time, often called *cutoff time*.

2.4.1.1 Algorithm Runtime

Viewed from the perspective of AS it might be tempting to believe that the time until a solution to a problem is found, is equivalent to the time the selected algorithm runs, i.e. algorithm runtime, and thus one should try to optimize algorithm runtime. However, this is not the case. In fact, under certain conditions, the time until a solution is found is equivalent to the runtime of the algorithm selector selecting the algorithm.

To understand this, let us take a closer look at the properties of the candidate algorithms $\mathcal{A}$. More precisely, consider the property of *completeness*. We call an algorithm *complete*, if (1) it terminates for all problem instances from $\mathcal{I}$ and (2) yields a valid solution to the corresponding instance as a result upon termination. If the algorithm does not satisfy these two conditions, we call it *incomplete*.

Now, suppose that we are given an instance $i \in \mathcal{I}$ and have a selector s, selecting algorithm $s(i) = a \in \mathcal{A}$. If a is incomplete, the runtime of a is obviously not equivalent to the time until a solution is found, as a might very well not output a solution upon termination or it might not terminate at all. In the first case, algorithm a clearly has a runtime, but the time until a solution is found is undefined. In the second case, both quantities are undefined. But even if a is complete, the runtime of a is not equivalent to the time until a solution is found, because

1. the algorithm selector s might require the computation of instance features $\boldsymbol{f}_i$ before being able to make a selection, which requires time to compute; and

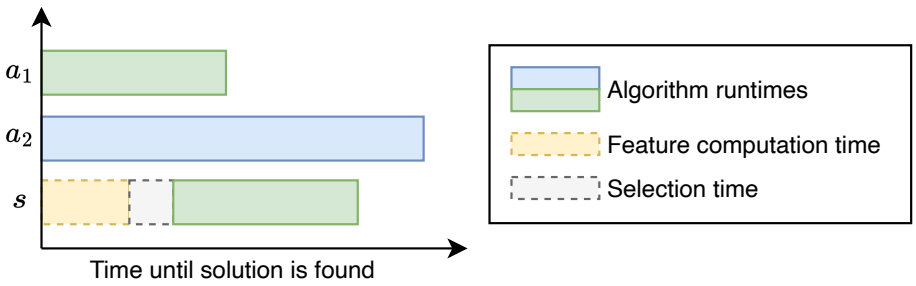

Fig. 2.9: This figure depicts the time until a solution is found of two complete algorithms a_1, a_2 and an algorithm selector s. In this example, the algorithm selector first computes features and then selects algorithm a_1. As one can see, the time until a solution is found associated with the algorithm selector s is larger than simply running the selected algorithm a_1 due to the feature computation time and the time the actual selection takes. Note that the latter is mostly negligible, but for visualization purposes, we depicted it here much larger than it would normally be.

2. the selection process itself does take time as well.

Correspondingly, if a is complete, the time until a solution is found is equivalent to the runtime of the algorithm selector s, if one assumes that the selector actually runs the selected algorithm at the end, as depicted in Figure 2.9. Moreover, algorithm runtime on its own can be trivially optimized by always selecting an algorithm, which instantly terminates without providing any solution as output.

Altogether, algorithm runtime on its own is not a good loss function, as

1. it does not consider whether a feasible solution to the problem has been generated by the selected algorithm,

2. it does not account for the computation of instance features which are often required by the algorithm selector; and

3. it does not account for the time the selection process itself requires.

While Item 2 can be accounted for by simply adding the feature computation time to the algorithm runtime and Item 3 is often negligible in practice, Item 1 requires explicit consideration in the loss function.

In practice, even if a is complete and hence eventually returns a solution to a given instance, there are often no guarantees or upper bounds on the runtime of

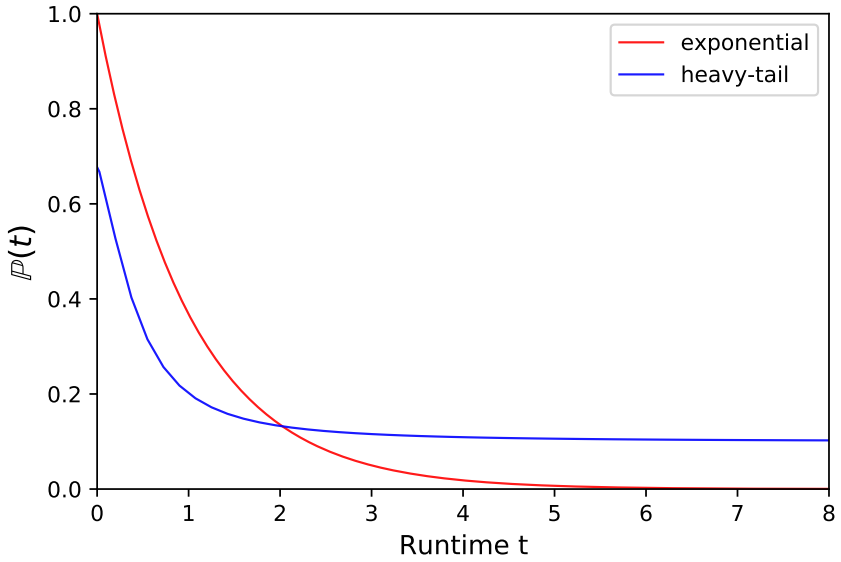

Fig. 2.10: Visualization of a heavy-tail runtime distribution (blue) in comparison to an exponential runtime distribution (red). Roughly speaking, as the name suggests, a heavy-tail distribution has a tail, which is heavier than the one of the exponential distribution.

an algorithm in AS problems. Correspondingly, a might run extremely long and potentially not finish within hours, years, or even decades. This behavior can be formalized by the observation that constraint satisfaction algorithms often exhibit so-called *heavy-tail* runtime distributions [GSC97]. Roughly speaking, this means that while an algorithm might solve some instances extremely fast, it might run extremely long on a subspace of $\mathcal{I}$. Figure 2.10 illustrates this phenomenon.

As a consequence, algorithms are usually run under a timeout, a so-called *cutoff time* C, and are forcefully terminated if they exceed this cutoff time. Naturally, this leads to the same problems as if a was incomplete, i.e. the instance we are originally trying to solve remains unsolved — a situation that should be avoided.

2.4.1.2 Number of Unsolved Instances

One way to overcome the problem of potentially unsolved instances that one faces when considering algorithm runtime as a loss function is to additionally track the number of such unsolved instances. For this purpose, one can define an indicator

loss function, which evaluates to 1, if no feasible solution is returned, and to 0, if a feasible solution is returned, i.e.

$$l_{unsolved}(i,a) = \begin{cases} 1 & \text{if } a \text{ does not return a solution to instance } i \text{ (until a cutoff)} \\ 0 & \text{else} \end{cases} \tag{2.36}$$

Then, in order to assess the loss of an algorithm selector s on a set of instances I, such a loss function can be used to count the number of unsolved instances by summation over these:

$$\mathcal{L}_{unsolved}(I,s) = \frac{1}{|I|} \sum_{i \in I} l_{unsolved}(i, s(i)) \tag{2.37}$$

Similarly to the algorithm runtime, the number of unsolved instances can be trivially optimized by always selecting a complete algorithm (in the worst case, a brute-force algorithm), which is guaranteed to eventually solve an instance, and thus, it should only be considered in combination with the algorithm runtime (including the feature computation times) yielding a multi-objective optimization problem. In fact, those two loss functions can even be contradicting as the two trivial optimization options we outlined show.

2.4.1.3 Penalized Average Runtime

Instead of directly tackling this multi-objective optimization problem, most existing AS research focused on constrained optimization problems falls back to an adapted algorithm runtime, which is penalized in case of unsolved instances, called the penalized average runtime (PAR).

The idea underlying the penalized average runtime is to penalize those algorithm runs, which did not solve an instance, i.e. which did not yield a solution until the cutoff time C (either because they failed or did not terminate). Let

$$l_{prK}(i,a) = \begin{cases} l_{runtime}(i,a) & \text{if } l_{runtime}(i,a) \leqslant C \\ K \cdot C & \text{else} \end{cases} \tag{2.38}$$

be the *K-penalized runtime* of algorithm a on instance i where $K \in \mathbb{N}$ controls the degree of penalization. Then, the *K-penalized average runtime* (PARK) of a selector s is defined as

$$\mathcal{L}_{PARK}(I, s) = \frac{1}{|I|} \sum_{i \in I} l_{prK}(i, s(i)) \ . \tag{2.39}$$

The most common choice for K is $K = 10$ resulting in a very strong penalty in case an algorithm did not solve an instance.

Unfortunately, the PARK cannot be reasonably aggregated over different sets of instances with potentially different runtime distributions and different cutoffs C due to large variances within the corresponding PARK scores. To mitigate this issue, one usually defines the *normalized* PARK score as

$$\mathcal{L}_{nPARK}(I, s) = \frac{\mathcal{L}_{PARK}(I, s) - \mathcal{L}_{PARK}(I, s^*)}{\mathcal{L}_{PARK}(I, SBS) - \mathcal{L}_{PARK}(I, s^*)} \ , \tag{2.40}$$

which is normalized with respect to the oracle s^* and SBS performance. Successful algorithm selectors should achieve an nPARK score below 1 as 1 corresponds to the performance of the SBS, values between 0 and 1 correspond to an improvement beyond the SBS, and a value of 0 corresponds to oracle performance.

Although variants of the PARK are arguably the most prominent loss functions to assess the quality of an algorithm selector in the algorithm selection literature [Lin+15; Bis+16; Tor+20a], the PARK has two main downsides. First, the concrete choice of the K is hard to make and thus rather arbitrary. While it is obvious that large choices of K coincide with a large penalty for timeouts, it is much less clear how a concrete requirement, for example, regarding the relative amount of timeouts, maps to a specific K [GLR22]. Second, simplifying the underlying multi-objective problem discussed earlier to an optimization of the PARK loss function is a rather crude approach in the form of a scalarization with all the potential disadvantages [ED18].

A potential solution to these problems is to abandon the simplification to a single-objective optimization problem altogether and directly tackle the underlying multi-objective optimization [Deb14] problem and then compare algorithms (and thus selectors) based on the notion of Pareto dominance and dominated hypervolume as suggested by Bossek and Trautmann [BT18]. However, as PARK is still the most dominant loss function used in the literature, most experimental parts of this thesis are based on evaluations of PARK.

2.4.2 Algorithm Selection Loss Functions for Optimization Problems

The concept of a constrained optimization problem is a generalization of the constrained satisfaction problem, where solutions satisfying the constraints are no longer necessarily equally preferred. Instead, an objective function $o : \mathcal{X} \longrightarrow [0, 1]$ assigning an objective value to a given solution is provided[8]. Then, the goal is to find a feasible solution maximizing the objective function o.

2.4.2.1 Solution Quality

While one is mostly interested in optimizing the time until a solution is found by the selected algorithm in the case of constraint satisfaction problems, one can alternatively optimize a measure based on the quality of the solution, i.e. its objective function value according to o, produced by the selected algorithm in the case of optimization problems. Nevertheless, even in cases where solution quality is optimized, algorithms can (and in fact often are), in principle, be run with a certain cutoff time. However, in practice, most optimization algorithms produce intermediate solutions during their run and often have a solution ready quite early in the process, such that unsolved instances can be neglected as they mostly do not exist. Correspondingly, solution quality is often the most important quantity to be optimized when considering constrained optimization problems in the context of algorithm selection.

In concordance with the previous notation, one can define solution quality in terms of a loss function as

$$l_{quality}(i, a) = 1 - o(a(i)) \tag{2.41}$$

where $a(i)$ denotes the solution obtained from applying algorithm a on instance i. Equivalently, the loss of a selector can then be defined as

$$\mathcal{L}_{quality}(I, s) = \frac{1}{|I|} \sum_{i \in I} l_{quality}(i, s(i)) \ . \tag{2.42}$$

[8] Note that we assume for simplicity that the objective function o maps to the unit interval. In principle, it could also map to other domains such as $\mathbb{R}$.

2.5 Instance Features

As mentioned earlier, a usual assumption associated with the AS problem is access to a feature map $f : \mathcal{I} \longrightarrow \mathbb{R}^d$ which, given an instance $i \in \mathcal{I}$ generates a $d-$dimensional feature representation of that instance. Such a representation allows one to generalize over instances and, in particular, to generate instance-specific algorithm selection suggestions on *unseen* instances using an underlying model, which depends on such features.

Before discussing different kinds of instance feature design methods (Section 2.5.2.1), we elaborate on the requirements of instance features in the following. Lastly, we shortly touch on feature preprocessing in Section 2.5.3. We note that although the design of such features is not the focus of this thesis, we want to provide the reader with a rough understanding of what kind of features are commonly used.

2.5.1 Requirements for Instance Features

In order to be useful in the context of algorithm selection, a (group of) feature(s) should fulfill a certain set of requirements. We note the following ones:

1. *Correlation*: The value of a feature for a specific instance should correlate with the performance of some or ideally all algorithms in $\mathcal{A}$.

2. *Computation time*: A feature should be fast to compute.

3. *Feature amount*: The total amount of features should be as small as possible.

4. *Complementarity*: Features should be complementary to each other in terms of their information.

5. *Domain independence*: Ideally, a feature should be problem domain-independent, such that it can be used on a large set of algorithmic problems.

Roughly speaking, Requirement 1, i.e. correlation, refers to the expressiveness of a feature wrt. to making good algorithm selection suggestions. If the value of a feature is completely uncorrelated with the performance of the algorithms in $\mathcal{A}$

it only contributes noise and should thus be removed. Consider, for example, a SAT problem where instances are named according to a 12-digit number drawn uniformly at random. Obviously, as long as the number is truly randomly drawn as described, a feature consisting of the instance number is not correlated with the performance of any algorithm and, thus, is of no use for selection suggestions. On the contrary, a feature describing the number of clauses and the number of literals is correlated with the empirical hardness of an instance and correspondingly the performance of many SAT solvers [Nud+04b] and thus might indeed be a good feature.

Requirement 2, i.e. computation time, is of a rather practical nature. Regardless of whether satisfaction or optimization problems are considered as underlying algorithmic problems, runtime often plays an important role in practice. Thus, algorithm selection suggestions are often required to be performed quickly such that the chosen algorithm can be run as soon as possible. Accordingly, the computation of a feature needs to be rather fast. This is especially important in satisfaction problems, where the given loss function l is often based on the runtime of the chosen algorithm in one way or the other and includes the time the algorithm selection process requires. Naturally, the overhead induced by the instance-specific algorithm selection process compared to simply running the SBS on all instances should be less than the difference in runtime between the chosen algorithm and the SBS (at least on average). Thus, it is crucial to include the feature computation times when evaluating how a certain AS approach performs.

Similarly, Requirement 3, i.e. feature amount, is also rather practically oriented. Firstly, a large set of features should be avoided as even when the features on their own are fast to compute, the sum over computation times of the features can quickly become non-negligible. Secondly, training data for algorithm selection is mostly rather sparse as often, there exists only the evaluation of a single algorithm (instead of all algorithms) on an instance in the training data. Consequently, complex machine learning models should only be used carefully to avoid the curse of dimensionality [Bac17]. Lastly, the larger the set of features, the larger the prediction time of the machine learning model used as part of the AS approach, and thus, the larger the overhead induced by AS becomes.

Along the same lines, Requirement 4, i.e. complementarity, is meant to describe that features should not share redundant information and, in particular, should not be correlated to each other. This is desirable since instance features, which are correlated with each other, tend to be problematic for some machine learning

Tab. 2.2: Tabular comparison of the requirements imposed upon good instance features by this work and by Kerschke et al. [Ker+19].

This work	Kerschke et al. [Ker+19]
Correlation	Informative
Computation time	Cheaply computable
Feature amount	—
Complementarity	Complementary
Domain independence	Generally applicable
—	Interpretable

models [Ali10] and, even more importantly, they unnecessarily increase the number of features leading to the same problems discussed above.

Lastly, Requirement 5, i.e. domain independence, concerns the re-usability of feature designs. Unfortunately, most features found in algorithm selection are problem dependent as they are carefully designed for a particular problem in order to fulfill Requirement 1 (correlation) [Ker+19]. As a consequence, they are often not easily transferable to a new problem class, which consequently requires a new set of domain experts to design a new set of features. Unfortunately, this design process is extremely cumbersome and also expensive due to its need for domain experts. Thus, it constitutes a research problem on its own [Ker+19]. A possible remedy for this problem lies within the idea of automated feature generation/extraction [Bra+22] — a process, which automatically tries to extract or generate features from a raw representation of an instance without the need for a domain expert. While this process yields domain-dependent features, the process itself is at least to some degree domain-independent and thus more broadly applicable.

We note that the set of requirements listed at the beginning of this section slightly deviates from the requirements listed by Kerschke et al. [Ker+19]. While Kerschke et al. [Ker+19] also list equivalent notions of our requirements 1 (correlation), 2 (computation time), 4 (low inter-feature correlation) and 5 (domain independence), we deviate in the remaining ones. In addition to the requirements just given, Kerschke et al. [Ker+19] note that features should be interpretable [DLH19]. We believe that interpretability of instance features is not necessarily required as this highly depends on the application. There certainly are applications where interpretability and correspondingly, explainable artificial intelligence [DBH18] methods can be beneficial in the context of AS. However, interpretability is not necessarily required to select the best-performing algorithm. A visual representation of the discrepancies in the requirements between Kerschke et al. [Ker+19] and this work,

Tab. 2.3: Overview of the requirements fulfilled by different kinds of instance features. A ✓ symbol indicates that the requirement is well fulfilled, a ◯ symbol indicates that it is somewhat fulfilled, whereas a ✗ symbol indicates that the requirement is not fulfilled. We intentionally left the assessment blank (?) for some requirements as this depends on the actual approach configuration.

Requirements	Instance feature kind		
	Syntactic	Probing	Deep learning-based
Correlation	◯	✓	✓
Computation time	✓	✗	✓
Feature amount	?	?	?
Complementarity	?	?	?
Domain independence	✗	✗	◯

including differences in the notions used to describe the requirements, can be found in Table 2.2.

2.5.2 Different Kinds of Instance Features

We distinguish between three kinds of instance features, which are commonly used in AS. First, we elaborate on syntactic instance features, which are often very domain-dependent and thus cannot be easily transferred across problem classes. Second, we discuss so-called probing features, which are based on statistics gained from very short algorithm runs. Third and last, we summarize work on automatically generated instance features based on deep learning.

For each such category of features, we also discuss whether they adhere to the requirements of instance features mentioned in Section 2.5.1. A visual overview of this assessment can be found in Table 2.3.

2.5.2.1 Syntactic Instance Features

As mentioned several times by now, handcrafted instance features are usually carefully designed by domain experts and correspondingly domain-dependent. While it is hard to give a general overview due to their domain-dependent nature, they are generally based on statistical quantities regarding the syntactic nature of an instance [OMa+08]. Examples include the number of decision variables of the instance

and their corresponding domains [Bis+16]. In addition, information extracted from structures computed from an instance (e.g. a constraint graph in the case of a constraint satisfaction problem [Gen+10]) are often used [Bis+16]. Due to the wide range of such instance features, we refrain from an excessive survey of corresponding work and refer the interested reader to Kerschke et al. [Ker+19].

In the field of meta-learning [Van18], which is concerned with learning about machine learning algorithm performance (for more details, see Section 2.2.2), the same kind of (meta-) features are called statistical (meta-)features.

Regarding the requirements of instance features (Section 2.5.1), syntactic ones can adhere to Requirement 1 (correlation), if correctly designed [KT19]. While most of them can be computed quite quickly such that they also fulfill Requirement 2 (computation time), the classical approach often involves using a large set of syntactic features to begin with, which is then reduced using feature selection techniques (cf. Section 2.5.3). Unfortunately, due to their domain dependence, most syntactic features inherently violate Requirement 5, as they cannot be easily transferred to other problem classes.

2.5.2.2 Probing Instance Features

Probing features refer to features, which are extracted from the trajectory obtained from a short run of an algorithm (possibly, but not necessarily from $\mathcal{A}$). Corresponding examples include the number of nodes explored up to a certain point in time in a heuristic search algorithm or the amount of pruned search nodes [Bis+16].

When considering continuous blackbox optimization as an underlying problem in AS, probing features are often called exploratory landscape analysis (ELA) [Mer+13] features. ELA is a set of techniques concerned with characterizing the blackbox function to optimize using a set of features ranging from simple sampling strategies to more complex information extracted from algorithm runs. ELA features have been shown to be quite useful in the context of AS [Bis+11; KT19]. Once again drawing the connection to meta-learning, similar kinds of features are called landmarking (meta-) features in that field.

Due to their high correlation with algorithm performance (Requirement 1), they are very frequently used in practice [Nud+04b; OMa+08; Xu+08; Hut+14]. Unfortu-

nately, they are often more complicated to compute than syntactic ones and thus also require more computation time. For this reason, they are sometimes also computed based on a timeout, which possibly results in missing feature values, if the timeout is reached, and the desired quantity is not computable using the acquired information. Although less domain-dependent than syntactic features, probing features still need to be adjusted to the corresponding problem class. Ideally, they are also adjusted based on the set of algorithms $\mathcal{A}$ involved, such that there are features specific to each of the algorithms.

2.5.2.3 Deep Learning-Based Instance Features

As our discussion so far and Table 2.2 illustrate, both syntactic and probing features are domain-dependent. In fact, the vast majority of existing work on algorithm selection (including the work presented in this thesis) is based on domain-dependent instance features, which inherently violate Requirement 5, i.e. domain independence, as also noted by Kerschke et al. [Ker+19]. In the last years, deep learning approaches, which can automatically learn complex features from rather raw input data [Ben12], have been leveraged to alleviate this problem.

Approaches going in this direction mostly follow the overall principle depicted in Figure 2.11. Instead of computing instance features using a domain-dependent feature function, they convert a raw instance $i \in \mathcal{I}$ into a representation r_i, which can be fed into a neural network. In most cases, this representation is either an image as input for a convolutional neural network (CNN) [LB95] or a series of numbers for online problems as input for a recurrent neural network (RNN) [Sch15]. This network is then supposed to predict the best-performing algorithm given the representation of the raw instance, i.e. it solves a classification task. The training of these networks works in a supervised manner based on classification training data

$$\mathcal{D}_{train} = \{(r_i, a_i^*) \mid i \in \mathcal{I}_D\} \tag{2.43}$$

consisting of raw representations r_i of the training instances $i \in \mathcal{I}_D$ and the algorithm $a_i^* \in \mathcal{A}$ performing best on the corresponding instance i [9].

[9] Depending on the concrete work, the label of an instance can also be of other nature, e.g., a one-hot encoded bit-vector inherently defining the associated algorithm by a 1 as in [Pra+21].

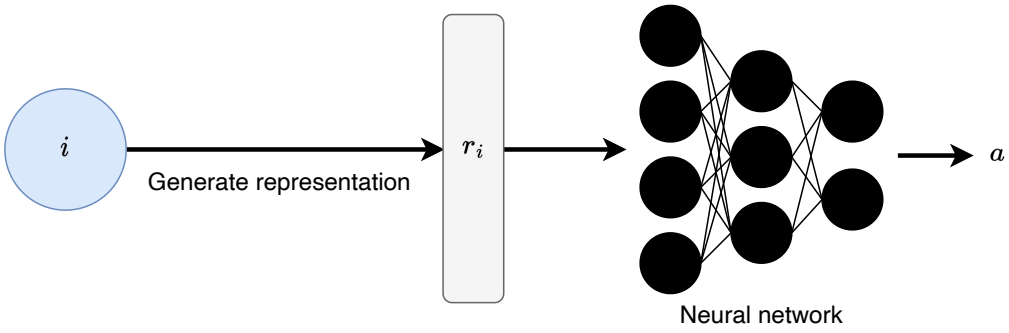

Fig. 2.11: Visualization of the working principle underlying AS approaches based on automated instance feature generation. In order to avoid computing instance features prior to performing algorithm selection, the raw instance is transformed into a representation r_i, such as an image, which can be fed into a neural network. Based on the input representation of the instance, the neural network outputs an algorithm $a \in \mathcal{A}$ to be applied.

Approaches following this scheme mainly differ in the input instance representation fed into the neural network and the architecture of the neural network used to predict, which algorithm should be selected.

An overview of literature following the scheme described above is given in Table 2.4. Most of these works perform successful algorithm selection in the sense that they beat the SBS on the corresponding loss function considered in the work. Moreover, some perform on par with standard AS approaches based on domain-dependent features and some even outperform such standard methods. We believe that this is an impressive achievement, considering that most of such approaches adhere to requirement 5 and are thus much more versatile in comparison to standard approaches. Furthermore, this kind of approaches feature a relatively low computation time as they mainly require computing the raw input representation followed by a pass through the neural network to compute the actual features and make a selection.

On the downside, a reader taking a closer look at the work in Table 2.4 may wonder to what degree the approaches are really domain-independent in the sense that no domain expert is required. While some [Lor+16] leverage a very general approach to represent instances by converting a text file to an image, the work by Seiler et al. [Sei+20] is specifically tailored towards TSP instances, where cities are represented by dots in an image. In fact, Seiler et al. [Sei+20] even show that when integrating domain knowledge into the image generation process, the performance of the overall approach can be increased.

Tab. 2.4: Overview and categorization of literature focusing on deep-learning-based automated instance feature generation.

Work	Problem	NN input repr.	Network type	Better than SBS ?	Better than std. AS ?
[Lor+16]	SAT, CSP	Image	CNN	✓	✗
[SB17]	Path finding in video games	Image	CNN	✓	?
[Sie+19]	Planning	Image	CNN	?	?
[ASH19; ASH22]	Online bin packing	Sequence	RNN	✓	✓
[Sei+20]	TSP	Image	CNN	✓	≈
[Pra+21]	Single-objective blackbox optimization	Image	CNN	✓	≈
[Zha+21]	TSP	Image	CNN	✓	✓

In the same spirit, a reader may wonder whether the notion of *feature-free* used by some of the approaches mentioned above to describe themselves is a good fit. As discussed, at least a raw representation of an instance is required as input for the corresponding neural network, which in itself is a feature representation. Correspondingly, we avoid the term in this work.

Moreover, while these works are certainly promising, a comprehensive study applying a general, possibly deep learning-based, instance representation technique as part of an AS approach over a wide range of different problems is, to the best of our knowledge, still missing. The study covering the widest range of problems was performed by Loreggia et al. [Lor+16], which covers SAT and CSP problems.

2.5.3 Feature Preprocessing

As discussed earlier, most instance feature representations violate at least one of the requirements listed in Section 2.5.1, and correspondingly, feature preprocessing can be useful to alleviate this situation at least to a certain degree. Most notably, imputation of missing feature values is often required in practice since instance features are often computed under a timeout as some of them, e.g. probing features (cf. Section 2.5.2.1), are quite time-intensive to compute. Correspondingly, if a timeout is hit during the computation, a feature value might be missing. Furthermore, as standard in machine learning, preprocessing steps can improve the overall

performance of an algorithm selector as, for example, demonstrated by Hutter et al. [HHL13], Amadini et al. [Ama+15], Bischl et al. [Bis+16], and Heins et al. [Hei+21].

As a particularly useful preprocessing technique in AS, feature selection [GE03] has the potential to reduce the overhead incurred by AS, if fewer features have to be computed for an instance. In fact, even when largely reducing the set of instance features, the (relative) performance of an algorithm (compared to a set of others) can still be sufficiently estimated [RH09; KM11; HHL13; Faw+14; Bis+16]. On the contrary, there also exist cases where feature selection does not seem beneficial [Kot+15]. Naturally, possible improvements evoked by feature selection, among other things, also depend on the machine learning model underlying the used AS approach.

2.6 ASlib: The Algorithm Selection Library

The ASlib [Bis+16] is a curated collection of AS scenarios featuring different problem domains [10] and is the de-facto standard benchmark for algorithm selection approaches [11]. Its format was designed in order to unify different formats for publishing AS scenarios, which have been used in the literature before.

An ASlib scenario consists of the following elements:

- A set of instances of an algorithmic problem, which is fixed across the scenario.
- A set of computed instance features, usually grouped by the procedure computing them, and their associated computational costs for each instance. In addition, it can also contain information about the computation status, e.g., failure or success, for each feature group.

[10] The concrete algorithmic problems included in ASlib are: answer set programming (ASP), bayesian network structured learning (BNSL), container premarshalling problem (CPMP), CSP, SAT, subgraph isomorphism (SGI), maximum satisfiability problem (MAXSAT), mixed integer programming (MIP), machine learning (ML), quantified boolean formula (QBF), TSP, traveling thief problem (TTP).

[11] The current set of scenarios can be found at `https://github.com/coseal/aslib_data`

- A predefined cross-validation split across the instances. This allows to easily compare evaluation results across papers as long as they used the predefined splits and the same evaluation measures.

- A set of algorithms, which can solve the instances.

- One or multiple loss functions $l_1, \ldots, l_n$.

- Evaluations of the loss function(s) l for the associated algorithms on the instances. In almost all cases, these evaluations are computed under a timeout, such that some algorithms time out on some instances resulting in missing evaluations in those cases (cf. Table 2.5). In addition, it can also contain information regarding the algorithm run status, e.g., if it crashed or timed out.

- Additional information such as the algorithm cutoff time, feature cutoff time, possibly algorithm configurations, and memory cutoffs for both features and algorithms.

Table 2.5 gives an overview of the scenarios contained in ASlib. Note that the concrete set of scenarios used in the different chapters might differ due to different loss functions being considered or due to certain properties of the scenarios making them unappealing for the corresponding chapters. The concrete set of scenarios used in this thesis is defined in the corresponding evaluation section — often as part of the result table. Moreover, ASlib is continuously growing and the results shown in this thesis have been developed over the course of four years such that not all scenarios were available when some of the evaluations were performed.

Tab. 2.5: Scenarios and corresponding statistics contained in ASlib.

Scenario	Algorithmic problem	#Instances	#Algorithms	#Features	#Unsolved instances	Relative #unsolved instances	Relative #missing evaluations	Cutoff (s)	Loss
ASP-POTASSCO	ASP	1294	11	138	82	0.06	0.20	600	runtime
BNSL-2016	BNSL	1179	8	86	0	0.00	0.28	7200	runtime
CPMP-2015	CPMP	527	4	22	0	0.00	0.28	3600	runtime
CSP-2010	CSP	2024	2	86	253	0.12	0.20	5000	runtime
CSP-MZN-2013	CSP	4642	11	155	944	0.20	0.70	1800	runtime, solved
CSP-Minizinc-Obj-2016	CSP	100	22	95	0	0.00	0.28	1	obj, time
CSP-Minizinc-Time-2016	CSP	100	20	95	17	0.17	0.50	1200	PAR10
GLUHACK-2018	SAT	353	8	50	116	0.33	0.55	5000	runtime
GLUHACK-2018-ALGO	SAT	353	8	50	116	0.33	0.55	5000	runtime
GRAPHS-2015	SGI	5725	7	35	117	0.02	0.07	1.0×10^8	runtime
GRAPHS-2015-ALGO	SGI	5725	4	35	117	0.02	0.07	1.0×10^8	runtime
MAXSAT-PMS-2016	MAXSAT	601	19	37	45	0.07	0.39	1800	PAR10
MAXSAT-WPMS-2016	MAXSAT	630	18	37	89	0.14	0.58	1800	PAR10
MAXSAT12-PMS	MAXSAT	876	6	37	129	0.15	0.41	2100	runtime
MAXSAT15-PMS-INDU	MAXSAT	601	29	37	44	0.07	0.49	1800	runtime
MAXSAT19-UCMS	MAXSAT	572	7	54	132	0.23	0.37	3600	runtime
MAXSAT19-UCMS-ALGO	MAXSAT	572	7	54	132	0.23	0.37	3600	runtime
MIP-2016	MIP	218	5	143	0	0.00	0.20	7200	PAR10
OPENML-WEKA-2017	ML	105	30	103	0	0.00	0.00	1	pred. accuracy
OPENML-WEKA-2017-ALGO	ML	105	21	103	0	0.00	0.00	1	pred. accuracy
PROTEUS-2014	CSP	4021	22	198	456	0.11	0.60	3600	runtime
QBF-2011	QBF	1368	5	46	314	0.23	0.55	3600	runtime
QBF-2014	QBF	1254	14	46	241	0.19	0.56	900	runtime
QBF-2016	QBF	825	24	46	55	0.07	0.36	1800	PAR10
SAT03-16_INDU	SAT	2000	10	483	269	0.13	0.25	5000	PAR10
SAT03-16_INDU-ALGO	SAT	2000	8	483	269	0.13	0.25	5000	PAR10
SAT11-HAND	SAT	296	15	115	77	0.26	0.61	5000	runtime
SAT11-HAND-ALGO	SAT	296	11	115	77	0.26	0.61	5000	runtime
SAT11-INDU	SAT	300	18	115	47	0.16	0.33	5000	runtime
SAT11-INDU-ALGO	SAT	300	18	115	47	0.16	0.33	5000	runtime
SAT11-RAND	SAT	600	9	115	108	0.18	0.48	5000	runtime
SAT11-RAND-ALGO	SAT	600	8	115	108	0.18	0.48	5000	runtime
SAT12-ALL	SAT	1614	31	115	20	0.01	0.54	1200	runtime
SAT12-HAND	SAT	767	31	115	229	0.30	0.67	1200	runtime
SAT12-INDU	SAT	1167	31	115	209	0.18	0.50	1200	runtime
SAT12-RAND	SAT	1362	31	115	322	0.24	0.73	1200	runtime
SAT15-INDU	SAT	300	28	54	17	0.06	0.24	3600	runtime
SAT18-EXP	SAT	353	37	50	67	0.19	0.51	5000	runtime
SAT18-EXP-ALGO	SAT	353	37	50	67	0.19	0.51	5000	runtime
SAT20-MAIN	SAT	400	67	108	77	0.19	0.52	5000	runtime
TSP-LION2015	TSP	3106	4	122	0	0.00	0.10	3600	runtime
TSP-LION2015-ALGO	TSP	3106	4	122	0	0.00	0.10	3600	runtime
TTP-2016	TTP	9720	21	55	0	0.00	0.05	1	solution quality

3 Extreme Algorithm Selection: Generalizing Across Algorithms

In this chapter, we discuss a change to a hidden assumption associated with the instance-specific offline algorithm selection (AS) setting. In particular, one often assumes that the set of algorithms only contains a handful, i.e. up to tens of algorithms, whereas we consider the case that the size of the set of algorithms $\mathcal{A}$ is extremely large, i.e. hundreds to thousands. In line with the emerging topic of extreme classification [Ben+18], we dub this setting with a significantly larger set of algorithms extreme algorithm selection (XAS) [1]. After explaining the differences to the standard offline AS setting in detail (Section 3.1), we elaborate on why the change of the size of the set of algorithms makes standard algorithm solutions impractical (Section 3.2) and how to adapt them in order to generalize across algorithms mitigating those problems (Section 3.3). In particular, we suggest not only representing instances but also algorithms, by features to enable such a generalization across algorithms — we call this a dyadic feature representation. We verify these claims in an experimental case study based on a benchmark AS scenario created for this work (Section 3.4), before putting our work into the context of existing work (Section 3.5).

For the remainder of this chapter, we assume no specific loss function. Moreover, we assume the formal setting of the standard algorithm selection problem as defined in Section 2.1.1 with some variations as discussed above.

The content presented in this chapter has been partly published in the form of a workshop paper [TWH19] and a conference paper [TWH20a].

[1] Note that we do not introduce this problem variant in the background chapter since its formal definition is equivalent to the offline AS problem, but it differs in certain aspects of hidden assumptions associated with the offline AS problem as we discuss in the following sections.

3.1 From Standard to Extreme Algorithm Selection

Hitherto practical applications of AS, as selecting a boolean satisfiability (SAT) solver for a logical formula, typically comprise candidate sets consisting of at most tens of algorithms, and this is also the order of magnitude that is found in standard AS benchmark suites such as ASlib (cf. Section 2.6). This is in contrast with the problems of algorithm configuration (AC) [Sch+22] and combined algorithm selection and hyperparameter optimization (CASH) [Tho+13] as considered in automated machine learning (AutoML) [HKV19], where the number of potential candidates is very large and potentially infinite [Tho+13; Feu+15; MWH18] (see also Section 2.2). Corresponding methods mostly rely heavily on computationally extensive search procedures combined with costly online evaluations of the performance measure to optimize for, since learning effective meta models for an instantaneous recommendation becomes very hard.

In this part of the thesis, we propose XAS as a novel setting in-between traditional AS and AC / CASH, which is motivated by application scenarios characterized by

- the demand for prompt recommendations in quasi real-time,
- an extremely large (though still finite) set of candidate algorithms.

An example is the scenario of "On-the-fly computing" [Hap+13], including "On-the-fly machine learning" [Moh+19] as one of its instantiations, where users can request online (machine learning) software services customized towards their needs. Here, users are unwilling to wait for several hours until their service is ready, but rather want to claim a result quickly. Hence, for providing a first version of an appropriate service, costly search and online evaluations are not affordable. As we will see, XAS offers a good compromise solution: Although it allows for the consideration of extremely many candidate solutions, and even offers the ability to recommend algorithms that have never been encountered so far, it is still amenable to AS techniques and avoids costly online evaluations.

In a sense, XAS relates to standard AS as the emerging topic of extreme classification (XC) [Ben+18] relates to standard multi-class classification. Similar to XC, the problem of learning from sparse data is a major challenge for XAS: For a single algorithm, there are typically only observations for a few instances, if at all.

Tab. 3.1: Overview of the characteristics of the problem settings we distinguish.

Characteristics/Setting	AS	XAS	AC	CASH
Size of $\mathcal{A}$	at most tens	extremely many	potentially infinite	potentially infinite
Training data	some missing	sparse	mostly not present	mostly not present
Online evaluations	no	no	yes	yes

3.1.1 Differences to Existing Problem Settings

More concretely, the XAS setting distinguishes itself from the standard AS setting (cf. Section 2.1.1) by two important properties. Firstly, we assume that the set of candidate algorithms $\mathcal{A}$ is *extremely* large. Thus, approaches need to be able to scale well with the size of $\mathcal{A}$. Secondly, due to the size of $\mathcal{A}$, we can no longer reasonably assume to have evaluations for most algorithms on most training instances. Instead, we assume that the training matrix spanned by the training instances and algorithms is only (very) sparsely filled. In fact, we might even have algorithms without any evaluations at all. Hence, suitable approaches need to be able to learn from very little data and to tackle the problem of zero-shot learning [Wan+19], i.e. estimate the performance of a previously unseen algorithm.

Similarly, the XAS setting differs from the AC and CASH settings in two main points. Firstly, dealing with real-valued hyperparameters, the set of (configured) algorithms $\mathcal{A}$ is generally assumed to be *infinite* in both AC and CASH, whereas $\mathcal{A}$ is still finite (even if extremely large) in XAS. More importantly, in both AC and CASH, one usually assumes having time to perform online evaluations of solution candidates at recommendation time. However, as previously mentioned, this is not the case in XAS, where instantaneous recommendations are required. Hence, the XAS setting significantly differs from the AS, AC, and CASH settings.

A summary of the main characteristics of these settings is provided in Table 3.1.

3.2 Standard Algorithm Selection Solutions in the Context of XAS

In order to elaborate on the weaknesses of standard, i.e. non-dyadic, solutions in the XAS setting, in the following, we shortly recall the main classes of AS solutions presented in Section 2.3 and discuss their weaknesses in the XAS setting.

3.2.1 Ranking and Regression Solutions

As mentioned in Section 2.3.2.2 and Section 2.3.2.3, both regression and ranking solutions often learn an algorithm-specific surrogate loss model

$$\widehat{l}_a : \mathcal{I} \longrightarrow \mathbb{R} \tag{3.1}$$

for each algorithm $a \in \mathcal{A}$ where the target domain either is a true loss estimate or rather a qualitative estimate as in the case of pointwise ranking models. The idea behind this model is to estimate the outcome of applying algorithm a to instance i in terms of the actual loss function l, i.e. $l(i, a)$. The final selection is then performed by selecting the algorithm with the lowest predicted loss according to the set of learned surrogate loss functions.

While this approach is easy and straightforward, it has a major disadvantage in the XAS setting. The number of surrogate models $\widehat{l}_a$ to be learned is extremely large as the number of algorithms is extremely large by virtue of the setting. Consequently, the approach has to keep track of a large number of models and also has to query each of these models in order to propose an algorithm selection when a new instance arrives. Moreover, since the training data is very sparse, the amount of datapoints available to train each of these surrogate loss models $\widehat{l}_a$ is potentially very small which may result in rather bad-performing models. Even worse, if there are no evaluations at all for an algorithm, which may very easily happen as discussed earlier, no model can be trained at all.

The situation is even more problematic for approaches, which, instead of learning a surrogate loss for each algorithm, learn models for subsets of algorithms such as SATzilla'11. SATzilla'11 learns one model for each distinct pair of algorithms (cf. Section 2.3.2.1), enlarging the number of models to be learned.

Overall, this makes both standard ranking and regression approaches, including corresponding hybrids (Section 2.3.2.5), rather unsuitable for the XAS setting.

3.2.2 Classification Solutions

In contrast to regression and ranking solutions, classification solutions (Section 2.3.2.1) do not learn an algorithm-specific surrogate loss, but rather directly try to predict the correct algorithm by learning a multi-class classification model of the form

$$s : \mathcal{I} \longrightarrow \mathcal{A} \tag{3.2}$$

and hence do not suffer from the same problems discussed above. However, as they are solving an extreme classification problem [Ben+18] in the XAS setting, they suffer from every problem associated with extreme classification, such as the problem of unbalanced datasets or tail labels. Tail labels are those kinds of labels, which are very rarely or maybe not at all present in the training (and also have a low probability in the underlying actual distribution). Correspondingly, they are inherently hard to predict correctly. Due to the very sparse training data of the XAS setting, the set of tail labels is rather large and those labels, which are present in the training data, might even occur only once or very rarely depending on the size of the dataset.

For this reason, we decided not to use them for the practical case study at the end of this chapter as we deem them unsuitable.

3.2.3 Collaborative Filtering Solutions

In contrast to the other three solution classes discussed so far, collaborative filtering (CF) solutions (Section 2.3.2.6) are inherently much better suited for the XAS setting due to their origin in the field of recommender systems, which deal with both large amounts of users (instances) and items (algorithms). As a short reminder, (model-based) CF solutions work by decomposing the (usually sparse) performance matrix $R^{|\mathcal{I}_D| \times |\mathcal{A}|}$, where an entry $R_{i,a} = l(i, a)$ corresponds to the performance of algorithm a on instance $i \in \mathcal{I}_D$ according to l, if this performance is known, and $R_{i,a} = ?$ otherwise. The rating matrix is decomposed into two matrices $U \in \mathbb{R}^{|\mathcal{I}_D| \times k}$ and $V \in \mathbb{R}^{k \times |\mathcal{A}|}$ w.r.t. some loss function $L(R, U, V)$, such that

$$R \approx \hat{R} = UV^\top , \tag{3.3}$$

where U can be interpreted as latent features of the instances and V as latent features of the algorithms. Here, k is the number of latent features. Thus, CF approaches inherently learn a dyadic feature representation and, in particular, a latent feature representation for each of the algorithms.

Overall, due to their origin in the field of recommender systems and their design with a large number of items (algorithms) in mind, CF solutions are very well suited for the XAS setting in terms of their capabilities and limitations as long as they are paired with a solution to the cold-start problem discussed in Section 2.3.2.6.

3.2.4 Clustering Solutions

Recall that clustering solutions (Section 2.3.2.5) can be seen as decompositional AS approaches leveraging other AS approaches on the formed clusters. Since they suffer from the same disadvantages as the approaches, which are used on the clusters, we do not consider them in the case study.

3.3 Exploiting a Dyadic Feature Representation

As previously discussed, very few of the standard approach classes to AS explained in Section 2.3 scale well to the XAS setting, as they do not generalize over algorithms; instead, the models are algorithm-specific and trained independently of each other. A natural idea, therefore, is to leverage explicit feature information on algorithms as well and correspondingly joint models.

More specifically, we propose to use a feature function $f^A : \mathcal{A} \longrightarrow \mathbb{R}^p$ representing algorithms as p-dimensional, real-valued feature vectors. Analogously to the instance feature function f defined in Section 2.1.1, we denote the corresponding feature representation, i.e. vector, of an algorithm a by $\boldsymbol{f}_a^A \in \mathbb{R}^p$.

Then, instead of learning one surrogate loss function per algorithm, the joint feature representation of an instance and an algorithm, allows us to learn a single joint model

$$\widehat{l} : f(\mathcal{I}) \times f^A(\mathcal{A}) \longrightarrow \mathcal{T}\,, \tag{3.4}$$

and hence to estimate the loss of a given algorithm a on a given instance i in terms of $\widehat{l}(\boldsymbol{f}_i, \boldsymbol{f}_a^A)$. Here, $\mathcal{T}$ either corresponds to $\mathbb{R}$ (in case a regression approach is sought) or to the set of all rankings over algorithms $\mathcal{R}(\mathcal{A})$ (in case of a ranking approach). Note that this is also a very natural idea in terms of the signature of the true loss function definition in Section 2.1.1. In line with the notion of the dyadic feature representation, in the following, we will denote such a pair of instance and algorithm as a dyad.

In the following, we show how both regression and ranking approaches can be adapted to incorporate the joint representation idea.

3.3.1 Regression

With the additional feature information at hand, we resolve to a single joint dataset comprised of examples $\left(\psi(\boldsymbol{f}_i, \boldsymbol{f}_a^A), l(i,a)\right)$ with dyadic feature information for all instances $i \in \mathcal{I}_D$ and algorithms $a \in \mathcal{A}$ for which a loss value $l(i,a)$ is known, instead of constructing one dataset per algorithm (and thus learning algorithm-specific loss surrogates). Here,

$$\psi : \mathbb{R}^d \times \mathbb{R}^p \longrightarrow \mathbb{R}^q \tag{3.5}$$

is a joint feature map that defines how the d-dimensional instance and the p-dimensional algorithm feature vectors are combined into a single feature representation of a dyad of length q. What is sought, then, is a (parametrized) surrogate loss function $\widehat{l}^{\boldsymbol{\theta}}_{regression} : \mathbb{R}^q \longrightarrow \mathbb{R}$, such that

$$\widehat{l}^{\boldsymbol{\theta}}_{regression}\left(\psi(\boldsymbol{f}_i, \boldsymbol{f}_a^A)\right) \tag{3.6}$$

is a (good) estimate of the loss of algorithm a on instance i, i.e. $l(i,a)$. Here, $\boldsymbol{\theta} \in \mathbb{R}^q$ is a real-valued, q-dimensional (weight) vector.

Obviously, the choice of ψ will have an important influence on the difficulty of the regression problem and the quality of the model (3.6). The regression task itself comes down to learning the parameter vector $\boldsymbol{\theta}$. In principle, this can be done exactly as in the non-dyadic case (cf. Section 2.3.2.2).

Note that this is a generalization of the approach used by SMAC [HHL; Lin+22] for predicting performances across instances in algorithm configuration. We allow for a generic joint feature map ψ and an arbitrary model for $\widehat{l}^{\boldsymbol{\theta}}_{regression}$, whereas SMAC

limits itself to a concatenation of features and trains a random forest for modeling $\hat{l}^{\boldsymbol{\theta}}_{regression}$. Once again, it is noteworthy that SMAC by itself is not applicable in the XAS setting, as it relies on costly online evaluations. However, we apply an approach similar to the model used for performance predictions by SMAC in the case study of this part of the thesis.

3.3.2 Ranking

A similar adaptation can be made for (label) ranking approaches discussed in Section 2.3.2.3. Formally, this corresponds to a transition from the setting of label ranking to the setting of *dyad ranking* (DR) as proposed by Schäfer and Hüllermeier [SH18].

The first major change in comparison to the ranking approach class discussed earlier concerns the training data, where the rankings π_i for instance i are now of the form

$$\psi\big(\boldsymbol{f}_i, \boldsymbol{f}^A_{a_{\pi_i(1)}}\big) \succ \ldots \succ \psi\big(\boldsymbol{f}_i, \boldsymbol{f}^A_{a_{\pi_i(|\mathcal{A}|)}}\big)\,. \tag{3.7}$$

Thus, we no longer represent an algorithm a simply by its label (a) in the ranking, but by the result of applying the feature map ψ to the dyadic representation composed of the instance features $\boldsymbol{f}_i$ and the algorithm features $\boldsymbol{f}^A_a$. Furthermore, as in the case of the dyadic regression idea presented earlier, we no longer learn one loss surrogate per algorithm, but a single joint model of the form (3.6) based on the dyadic feature representation.

3.3.2.1 Dyadic Plackett-Luce Model

We chose a Plackett-Luce (PL) [CHD10; SH18] model as a ranking model, which specifies a parameterized probability distribution on rankings over dyads, i.e. instance and algorithm pairs. In comparison to other models, e.g., the Mallows model [Mal57], the PL model is well-suited for our case as it can be rather easily learned when rankings are incomplete [CHD10] in the sense that they do not feature the complete algorithm set $\mathcal{A}$. In fact, this is quite likely in the XAS setting. In the context of ranking models, one often rather uses utility models instead of loss functions as this is much more intuitive regarding the computation of the probability of a ranking

as we will see shortly. For this purpose, instead of learning a surrogate loss function, we learn a parameterized, joint utility model

$$\widehat{u}^{\boldsymbol{\theta}}_{rank} : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}_{\geqslant 0} \ , \tag{3.8}$$

which estimates how well an algorithm is suited for a given instance. In principle, we could convert $\widehat{u}^{\boldsymbol{\theta}}_{rank}$ into a loss function, but we abstain from doing so for better understandability. It is noteworthy, that we do not try to make $\widehat{u}^{\boldsymbol{\theta}}_{rank}$ a good estimate of the actual (inverted) loss function l here, but rather want to learn a function, which allows to qualitatively rank between dyads.

We model the latent utility $\widehat{u}^{\boldsymbol{\theta}}_{rank}$ as a log-linear function

$$\widehat{u}^{\boldsymbol{\theta}}_{rank}(i, a) = \exp\left(\boldsymbol{\theta}^\top \left(\psi(\boldsymbol{f}_i, \boldsymbol{f}^A_a)\right)\right), \tag{3.9}$$

where $\boldsymbol{\theta} \in \mathbb{R}^q$ is once again a real-valued, q-dimensional (weight) vector, which has to be learned.

Using this latent utility function, the PL model specifies a probability distribution on rankings over the algorithms: given an instance $i \in \mathcal{I}$, the probability of a ranking $a_1 \succ \ldots \succ a_z$ over any subset $\{a_1, \ldots, a_z\} \subseteq \mathcal{A}$ is

$$\mathbb{P}(a_1 \succ \ldots \succ a_z \,|\, \boldsymbol{\theta}) = \prod_{n=1}^{z} \frac{\widehat{u}^{\boldsymbol{\theta}}_{rank}(i, a_n)}{\widehat{u}^{\boldsymbol{\theta}}_{rank}(i, a_n) + \ldots + \widehat{u}^{\boldsymbol{\theta}}_{rank}(i, a_z)} \,. \tag{3.10}$$

A probabilistic model of that kind suggests learning the parameter vector $\boldsymbol{\theta}$ via maximum likelihood estimation, i.e. by maximizing the likelihood function

$$\prod_{i \in \mathcal{I}_D} \mathbb{P}(\pi_i \,|\, \boldsymbol{\theta}) \tag{3.11}$$

associated with Equation 3.10; this approach is explained in detail by Cheng et al. [CHD10]. Hence, the associated loss function under which we learn is now of a probabilistic nature (the logarithm of the PL-probability). Once again, it no longer focuses on the difference between the approximated algorithm loss $\widehat{l}^{\boldsymbol{\theta}}_{regression}(i, a)$ as in the regression case and the true loss of an algorithm $l(i, a)$, but on the ranking of the algorithms with respect to l — putting it in the jargon of preference learning, the former is a "pointwise" while the latter is a "listwise" method for learning to rank [Cao+07].

As a concrete instantiation of the latent utility function $\widehat{u}^{\boldsymbol{\theta}}_{rank}$ we use a feed-forward neural network, where $\boldsymbol{\theta}$ represents its weights, which, as shown by Schäfer and Hüllermeier [SH18], can be learned via maximum likelihood estimation on the likelihood function implied by the underlying PL model. Note that the use of a neural network is of particular interest here, since it allows one to learn the underlying joint feature map ψ implicitly. Although both instance and algorithm features are simply fed as a concatenated vector into the network, it can recombine these features due to its structure and thus implicitly learn such a joint feature representation.

3.3.3 Advantages and Disadvantages of Dyadic Approaches

The dyadic feature representation allows to generalize both across instances and algorithms and thus also allows to rate an unknown algorithm as long as a feature representation of the algorithm can be computed. Furthermore, leveraging a joint feature map in order to combine the instance and algorithm features, enables one to learn a joint loss surrogate instead of algorithm-specific ones. Correspondingly, dyadic approaches overcome the scaling problems of standard AS methods in the XAS setting discussed in Section 3.2. In particular, the extremely sparse training data is less of a problem for dyadic approaches as all data can be used to train the joint model instead of splitting the already limited data onto multiple models.

On the downside, designing good algorithm features is a tedious task, and very little work leading in this direction has been done (cf. Section 3.5). Consequently, this is a considerable limitation of our proposed dyadic approach. For the empirical evaluation in the form of a case study, we will use the parameter values of the algorithm as features as explained in detail in Section 3.4.1. Furthermore, combining the instance and algorithm features in a reasonable way, i.e. designing a good joint feature map ψ, is also a non-trivial problem and thus also a limitation of dyadic approaches. In the case study, we demonstrate two forms of such joint feature maps — a simple concatenation and an inherently learned joint feature map.

3.4 Experimental Evaluation: A Case Study

In our experiments, we evaluate well-established approaches to AS as well as the proposed dyadic approaches in the XAS setting. More specifically, we consider the problem of selecting a machine learning classifier (algorithm) for a new classification dataset (instance) as a case study related to the "on-the-fly machine learning" scenario [Moh+19] [2]. To this end, we first generate a benchmark scenario[3] and then use this benchmark for comparison. The generated benchmark scenario as well as the implementation of the approaches, including detailed documentation, is provided on GitHub[4]. We used custom implementations of all approaches considered for the evaluation, although we used several libraries for some of their components. More details on this can be found in Section A.1 and the GitHub repository.

3.4.1 Benchmark Scenario

In order to benchmark the dyadic approaches presented above in the XAS setting, we consider the algorithmic problem of machine learning. More precisely, the algorithmic problem is to select a classification algorithm for an (unseen) dataset, corresponding to an instance in the AS jargon. That being said, when talking about an instance we do not refer to the instance being part of a classic machine learning dataset, but to the corresponding dataset itself.

The benchmark scenario is fully defined by a finite set of algorithms $\mathcal{A}$ for classification, a set of instances $\mathcal{I}$ corresponding to classification datasets, and a loss function l estimating the loss of the corresponding XAS method. All of these are described in the following.

3.4.1.1 Algorithms

We define the set of candidate algorithms $\mathcal{A}$ by sampling up to 100 different parameterizations of 18 classification algorithms from the machine learning library WEKA

[2] This is just one among many conceivable instantiations of the XAS setting, which is supposed to demonstrate the performance of the presented methods

[3] We chose the notion of a scenario in line with ASlib (cf. Section 2.6).

[4] `https://github.com/alexandertornede/extreme_algorithm_selection`

[Fra+05]. This yields a total of $|\mathcal{A}| = 1,270$ algorithms. The amount of parametrizations for each algorithm depends on the number and types of hyperparameters it features. We refer to the potential set of parametrizations drawn for a classification algorithm as the family of that algorithm. An overview of the algorithm families, their hyperparameters, and the number of instantiations contained in $\mathcal{A}$ is given in Table 3.2.

3.4.1.2 Instance Space

The instance space $\mathcal{I}$ is based on the OpenML CC-18 benchmarking suite[5] [Van+13], which is a curated collection of 71 classification datasets that are considered interesting from a model selection and hyperparameter optimization point of view. This property makes the datasets particularly appealing for the XAS benchmark scenario, as it ensures more diversity across the algorithms.

Accordingly, the total rating/performance matrix spanned by the algorithms and classification datasets in principle features $1,270 \cdot 71 = 88,900$ entries. An evaluation of each algorithm on each dataset results in filling $55,919$ of such entries with a loss value of the corresponding algorithm on the corresponding dataset, whereas the rest is unknown due to errors during the evaluation.

3.4.1.3 Loss Function

In the domain of machine learning, one is usually more interested in the generalization performance of an algorithm instead of its runtime. Therefore, the loss function l should assess the quality of the machine learning model produced by a machine learning algorithm a on instance i. To this end, we carry out a 5-fold cross-validation on the corresponding instance, i.e. machine learning dataset, and measure the mean accuracy across the folds.

We note that accuracy is not a loss function, but rather a performance measure. Nevertheless, one can be easily converted into the other.

[5] `https://docs.openml.org/benchmark/#openml-cc18` (Excluding datasets 554, 40923, 40927, 40996 due to technical issues.)

Tab. 3.2: The table shows the types of classifiers used to derive the set $\mathcal{A}$. Additionally, the number of numerical hyperparameters (#num.P), categorical hyperparameters (#cat.P), and instantiations (n) is shown.

Learner	0R	1R	BayesNet	DecisionStump	DecisionTable	IBk	J48	JRip	KStar	Logistic	LMT	MultilayerPerceptron	NaiveBayes	PART	REPTree	RandomForest	RandomTree	SVM
#num.P	0	1	0	0	1	1	2	2	1	1	2	2	0	2	3	3	4	1
#cat.P	0	0	2	0	3	3	6	2	2	0	5	6	2	2	2	2	4	2
n	1	30	12	1	45	89	100	100	99	100	100	100	3	91	100	99	100	100

As the measure of interest, accuracy is a reasonable though to some extent arbitrary choice. Note that in principle any other measure could have been used for generating the benchmark as well.

3.4.1.4 Instance Features

For the setting of machine learning, the instances are classification datasets and the corresponding feature representations are called meta-features as discussed in Section 2.5.2. We use a specific class of meta-features in order to represent the datasets, called *landmarkers*, which are performance scores of cheap-to-validate algorithms on the respective dataset. More specifically, we use all 45 landmarkers as provided by OpenML [Van+13], for which different configurations of the following learning algorithms are evaluated based on the error rate, area under the (ROC) curve, and Kappa coefficient: Naive Bayes, One-Nearest Neighbor, Decision Stump, Random Tree, REPTree, and J48. Hence, in total, we use 45 features to represent a classification dataset, i.e. an instance in the benchmark scenario.

3.4.1.5 Algorithm Features

The presumably most straightforward way of representing an algorithm in terms of a feature vector is to use the values of its hyperparameters. Thus, we describe each individual algorithm by a vector of their hyperparameter values.

Due to the way in which we generated the set of candidate algorithms $\mathcal{A}$, we can compress the vector sharing features for algorithms of the same type. For example, if

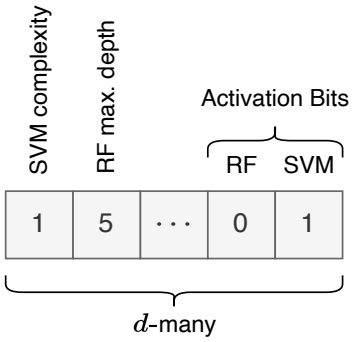

Fig. 3.1: Exemplary visualization of the algorithm feature vector concept. Each algorithm's hyperparameters are encoded in the first part of the vector whereas the last part contains an activation bit for each of the algorithms.

we consider multiple algorithms, which have a learning rate as a hyperparameter, we only need a single entry for the learning rate in the vector. Additionally, we augment the vector with a single categorical feature denoting the type of algorithm. Given any candidate algorithm, its feature representation is obtained by setting the type of algorithm indicator feature to its type, each element of the vector corresponding to one of its hyperparameters to the specific value, and other entries to 0. Categorical hyperparameters, i.e. features, are one-hot encoded yielding a total of 153 features to represent an algorithm. An exemplary visualization of this idea can be found in Figure 3.1.

Although simple, such a representation limits the applicability of our approach, since we cannot represent an arbitrary algorithm but only one that comes from any of the known families. Moreover, the representation is limited in terms of generalization across different algorithm families, as these are essentially represented by disjoint subvectors of the original vector if they do not share any hyperparameter. For the same reason, large parts of the vector will simply be 0 depending on the algorithm for which the representation was computed, which might yield a rather problematic behavior when trying to estimate the loss or performance of the algorithm based on its feature vector. Nevertheless, our experimental results show that even with such a simple representation, we can obtain good selection performance even under sparse training data.

3.4.1.6 Caveats

We would like to note that the problem underlying this benchmark scenario could of course be cast as an hyperparameter optimization (HPO)/AC or CASH problem. However, in order to instantiate the XAS setting we deliberately drastically limit the configuration possibilities such that a very large, but finite set of algorithms to choose from is created. We make the assumption that there is no time for costly online evaluations due to the on-the-fly setting (Section 3.1) and hence standard HPO/AC and CASH methods are not applicable. In a way, one can think of this benchmark scenario as a zero-shot HPO problem [Özt+22] with a very limited configuration space.

3.4.2 Baselines

Throughout the study, we employ various baselines to better relate the performance of the different approaches to each other and to the problem itself, i.e. assessing whether the more sophisticated approaches prove beneficial compared to simple and straightforward approaches.

All of the baselines are defined in a way to be able to create a ranking across algorithms instead of selecting only one to foster a more advanced comparison.

We define the following simple baselines:

`RandomRank` is a naive baseline choosing ranks for each algorithm at random.

`AvgPerformance` first averages the observed performance values across all training datasets for each algorithm. Then, it derives a statically predicted ranking, sorting the candidate algorithms according to their average performances. Observe that according to the definition, the highest ranked algorithm according to `AvgPerformance` is the single best solver (SBS).

`1-NN LR` is a nearest-neighbor-based label ranking approach which, given a new dataset, retrieves the training dataset with the smallest Euclidean distance to this new dataset based on their feature representations. The ranking associated with the respective training dataset is then returned as a prediction.

`2-NN LR` retrieves the two closest training datasets, averages the performances of each algorithm across these two datasets and returns the ranking implied by these average performance values.

Since a static ranking based on the average ranks of the algorithms across datasets is commonly used as another baseline in the standard AS setting, we note that we omit this baseline on purpose. This is because meaningful average ranks of algorithms are difficult to compute in the XAS setting, where the number of algorithms evaluated, and hence the length of the rankings of algorithms, vary from dataset to dataset.

In addition to the rather simple baselines above, we also compare against more sophisticated methods:

PAReg refers to the approach of learning a set of algorithm-specific loss function surrogates $\widehat{l}_a$, one for each algorithm a, through a regression technique as explained in Section 3.2.1. Here, we employ a random forest per algorithm. The ranking produced by `PAReg` for an instance i is then obtained by ranking all algorithms in $\mathcal{A}$ according to their estimated loss values $\widehat{l}_a(i)$ in increasing order. For those algorithms with no training data at all, we make `PAReg` predict an accuracy of 0, as recommending such an algorithm does not seem reasonable. Recall that it is not uncommon in the XAS setting that there is no training data for some algorithms.

Alors (REGR) / Alors (NDCG) [MS17] is a CF approach, which can deal with unknown instances by learning a feature map from the original instance to the latent instance feature space. It leverages the CF approach CoFi$^{\text{RANK}}$ [Wei+07] using the normalized discounted cumulative gain (NDCG) [Wan+13] as loss function $L(R, U, V)$. Since the NDCG is a ranking loss, it focuses on decomposing the rating/performance matrix R so as to produce an accurate ranking of the algorithms. More precisely, it uses an exponentially decaying weight function for ranks, such that more emphasis is put on the top and less on the bottom ranks. `Alors (REGR)` is a slight variation of Alors, which optimizes a regression loss instead of the NDCG.

3.4.3 Performance Metrics

We compute the following performance metrics measuring desirable properties of XAS approaches.

regret@k is the difference between the performance value of the best algorithm within the predicted top-k of algorithms and the actual best algorithm. The domain of regret@k is $[0, 1]$, where 0 is the optimum meaning no regret.

NDCG@k is a position-dependent ranking measure (*n*ormalized *d*iscounted *c*umulative *g*ain) to measure how well the ranking of the top-k ($k \leqslant |\mathcal{A}|$) algorithms can be predicted. It is defined as

$$\text{NDCG@}k(\pi, \pi^*) = \frac{\text{DCG@}k(\pi)}{\text{DCG@}k(\pi^*)} = \frac{\left(\sum_{n=1}^{k} \frac{2^{l(i,\pi_i(n))}-1}{\log(n+2)}\right)}{\left(\sum_{n=1}^{k} \frac{2^{l\left(i,\pi_i^*(n)\right)}-1}{\log(n+2)}\right)},$$

where i is a (fixed) instance, π is a ranking and π^* the optimal ranking, and $\pi_i(n)$ gives the algorithm on rank n in ranking π for instance i. The NDCG emphasizes correctly assigned ranks at higher positions with an exponentially decaying importance. It ranges in $[0, 1]$, where 1 is the optimal value.

Kendall's τ is a rank correlation measure. Given two rankings (over the same set of elements, i.e. algorithms) π and π', it is defined as

$$\tau(\pi, \pi') = \frac{C - D}{\sqrt{(C + D + T_\pi) \cdot (C + D + T_{\pi'})}} \tag{3.12}$$

where C/D is the number of so-called *concordant/discordant* pairs in the two rankings, and $T_\pi/T_{\pi'}$ is the number of ties in π/π'. Two elements are called a concordant/discordant pair if their order within the two rankings is identical/different, and tied if they are on the same rank. Intuitively, this measure determines on how many pairs the two rankings coincide. It takes values in $[-1, 1]$, where 0 means uncorrelated, -1 inversely, and 1 perfectly correlated.

3.4.4 Experimental Setup

In the following experiments, we investigate the performance of the different approaches and baselines in the setting of XAS for the example of the proposed benchmark scenario as described in Section 3.4.1.

We conduct a 10-fold cross-validation to divide the benchmark scenario into 9 folds of known and 1 fold of unknown benchmark scenario instances. In order to estimate how well the different approaches can deal with limited training data, we draw a sample of 25, 50, or 125 pairs of algorithms for every instance from the resulting set of known performance values under the constraint that the performance of the two algorithms is not identical. Thus, a maximum fill degree of 4%, 8% respectively 20% of the performance matrix is used for training, as algorithms may occur more than once in the sampled pairs. The sparse number of training examples is motivated by the large number of algorithms in the XAS setting. The assumption that performance values are only available for a small subset of the algorithms is clearly plausible here. Throughout the experiments, we ensure that all approaches are provided the same instances for training and testing, and that the label information is based on the same performance values. We note that the labels are not identical for all approaches, as, obviously, ranking approaches require other labels than regression approaches.

In the experiments, we compare two approaches leveraging a dyadic feature representation, which we derived in this chapter (Section 3.3) to the baselines described in Section 3.4.2. They internally fit either a random forest for regression (`DFReg` — cf. Section 3.3.1) or a feed-forward neural network as a dyad ranking model (`DR` — cf. Section 3.3.2). As a joint feature map ψ we use the simple concatenation of the instance and algorithm features for the dyadic approaches. As already mentioned, the neural network-based dyad ranking approach implicitly learns a more sophisticated joint feature map. However, `DFReg` only leverages this simple concatenation.

In contrast to the other methods, the ranking model `DR` is only provided the information which algorithm of a sampled pair performs better, as opposed to the exact performance value that is given to other methods. A summary of the type of features and label information used by the different approaches/baselines is given on the left side of Table 3.3.

Tab. 3.3: Overview of the data provided to the approaches and their applicability to the considered scenarios. Recall that f computes instance feature function and f^A computes algorithm features. An l in the label column indicates that the approach is trained on the loss function evaluations, whereas a π indicates that it is trained on rankings.

	Approach	f	f^A	Label		Approach	f	f^A	Label
Advanced	`Alors (REGR)`	✓	✗	l	Baselines	`RandomRank`	✗	✗	
	`Alors (NDCG)`	✓	✗	l		`AvgPrfm`	✗	✗	l
	`PAReg`	✓	✗	l		k-`NN LR`	✓	✗	l
	`DFReg`	✓	✓	l					
	`DR`	✓	✓	π					

Note that `DFReg` corresponds to the model underlying SMAC on an algorithmic level as discussed earlier. However, it is only trained once offline and not updated with costly online evaluations, which are not feasible in the XAS setting.

The test performance of the approaches is evaluated by sampling 10 algorithms for every (unknown) instance to test for and letting the approaches rank these. The comparison is done with respect to different metrics detailed further below, and the outlined sampling evaluation routine is repeated 100 times.

Statistical significance w.r.t performance differences between the best method and any other method is determined by a Wilcoxon signed rank sum test with a threshold of 0.05 for the p-value. Significant improvements of the best method over another one is indicated by $\bullet$.

3.4.5 Results

We visualize the results of the experiments described both in form of Table 3.4 and Figures 3.2 – 3.6. While the table displays the performance of the corresponding approach for the corresponding fill rate under a certain measure, there is one figure for each measure showing how the performance of the different approaches is affected by changes in the fill rate. However, the information displayed is in principle the same both for the table and the figures.

3.4.5.1 Performance of Dyadic Approaches

When considering Kendall's τ as a measure, either DR or DFReg performs best for each fill rate and even outperforms all other baselines significantly, meaning that they generate the best rankings across all considered approaches. The standard approach of training one model per algorithm to consider, i.e. PAReg, performs second best for all fill rates closely followed by the simpler baselines. Strikingly, both Alors variants perform very badly. Regarding the change in performance with an increasing fill rate, most approaches behave as expected and improve. However, the nearest neighbor approaches deteriorate and thus seem to be negatively impacted by the additional training data. Similarly, the DFReg approach decreases in performance for a fill rate of 8% compared to 4%, but improves again for a fill rate of 20%. While we could not find a concrete reason for this behavior, one has to keep in mind that the approach optimizes a regression loss and thus not necessarily generates a good ranking, but rather aims at predicting the algorithm performances as precisely as possible. Moreover, the NDCG variant of Alors does perform similarly for all fill rates.

Analyzing the approach behavior in terms of the NDCG measures, most of the trends discussed above for the Kendall's τ remain valid. Most strikingly, Alors (NDCG), which internally optimizes for the NDCG as a target measure, still yields rather bad results and even drops in performance for a fill rate of 8% compared to 4%.

Lastly, looking at the regret measure variants, other trends can be observed. First, although either DR or DFReg still yield the best performance for all fill rates, they do not outperform all other approaches significantly anymore. In fact, for a fill rate of 4%, the difference to the AvgPerformance baseline, performing very well, is very small but becomes larger with increasing fill rate. Once again, both Alors variants perform very badly and only slightly better than the RandomRank baseline indicating that the top elements of the ranking are not well chosen by Alors.

Overall, it becomes evident that the methods for standard algorithm selection tend to fail especially in the scenarios with only a few algorithm performance values per instance, although, depending on the considered measure, some of the baselines perform astonishingly well despite very little data. Moreover, Alors even fails to improve over simple baselines, such as AvgPerformance and k-NN LR. With an increasing number of training examples, PAReg improves over the baselines and also performs better than Alors, but never yields the best performance for any of the

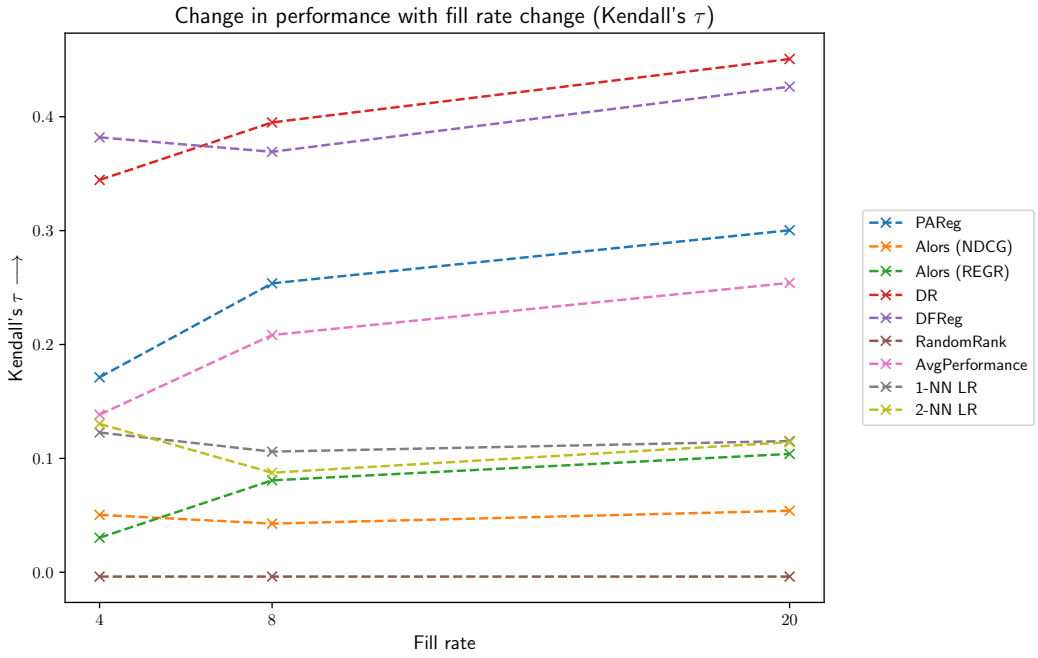

Fig. 3.2: Performance of different approaches for different fill rates in terms of Kendall's τ.

considered settings or metrics. We attribute the bad performance of `Alors` to its need to predict the latent instance features from the given meta-features and thus a potential error propagation. Moreover, it seems that the algorithm underlying the optimization of the NDCG, i.e. CoFi$^{\text{RANK}}$ does not perform very stable on our data. Independent of whether a regression loss or the NDCG is optimized, the final training loss achieved by `Alors` was often rather high. Nevertheless, the bad performance of `Alors` is very disappointing since CF based approaches are naturally very well suited for the XAS setting as discussed earlier. Note that despite the greatest care, we can, of course, not exclude that this performance might be caused by an error in the application of the CoFi$^{\text{RANK}}$ executable on our side.

In contrast to this, the proposed dyadic feature approaches clearly improve over both the methods for the standard AS setting and the considered baselines for all the metrics. Interestingly, `DFReg` performs best for the setting with only 25 performance value pairs, while `DR` has an edge over `DFReg` for the other two settings. We attribute this to the neural network used as a model in the PL model within the `DR` approach, which seems to require more data than the 25 pairs. Still, the differences between the dyadic feature approaches are never significant, whereas significant improvements can be achieved in comparison to the baselines and the other AS approaches.

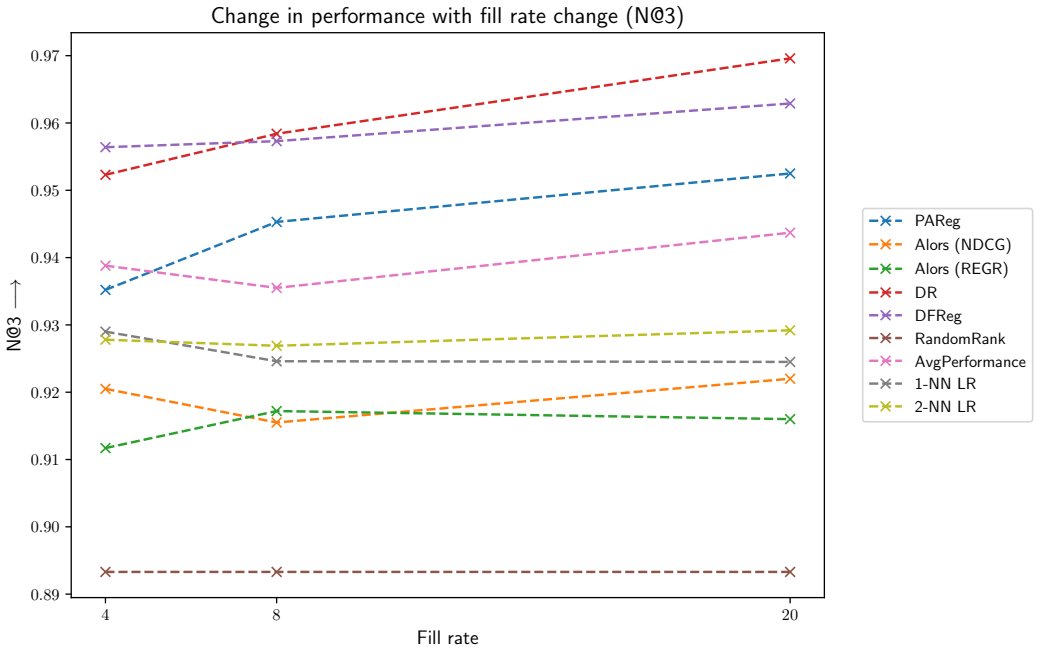

Fig. 3.3: Performance of different approaches for different fill rates in terms of NDCG@3.

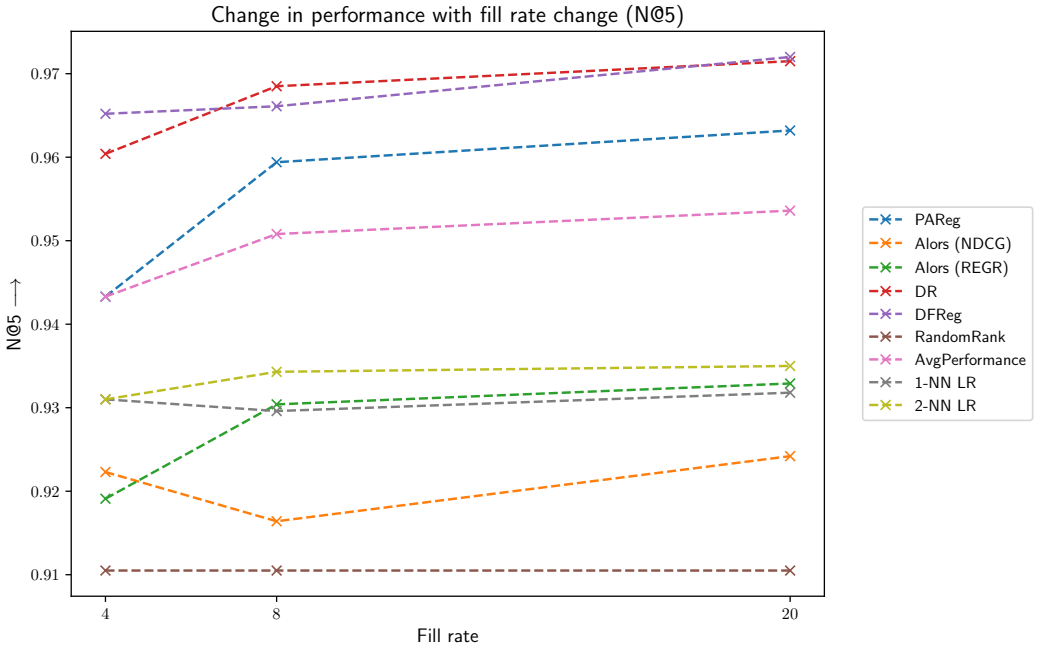

Fig. 3.4: Performance of different approaches for different fill rates in terms of NDCG@5.

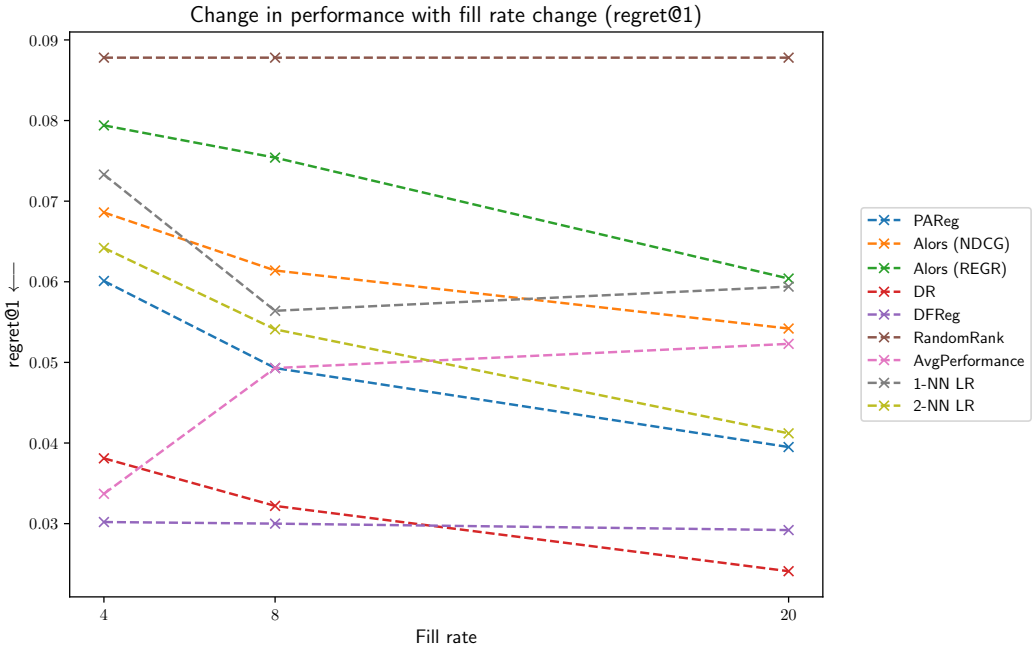

Fig. 3.5: Performance of different approaches for different fill rates in terms of regret@1.

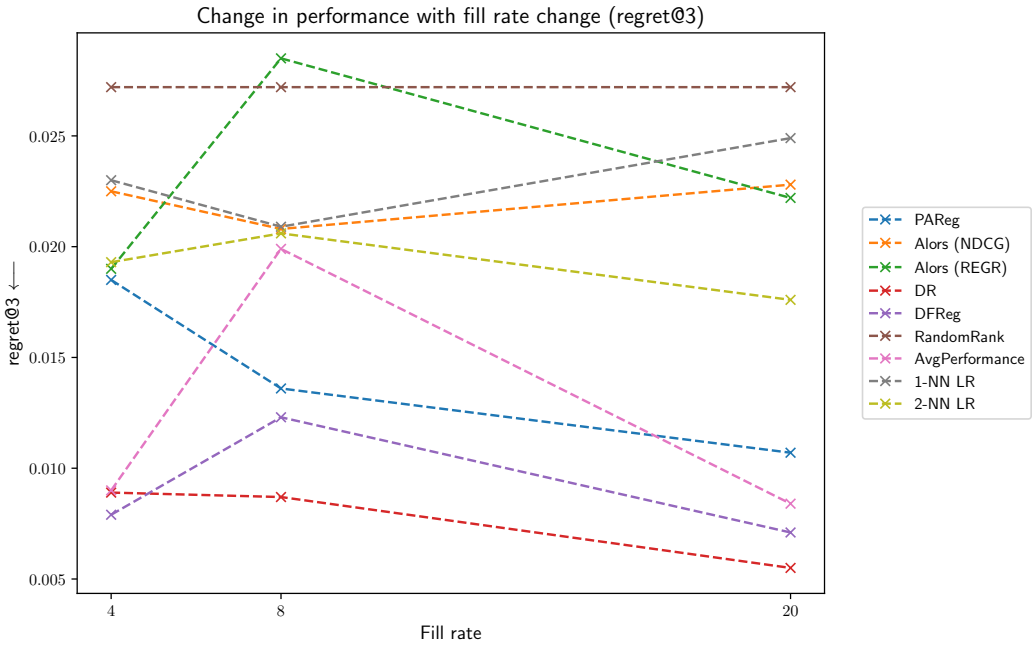

Fig. 3.6: Performance of different approaches for different fill rates in terms of regret@3.

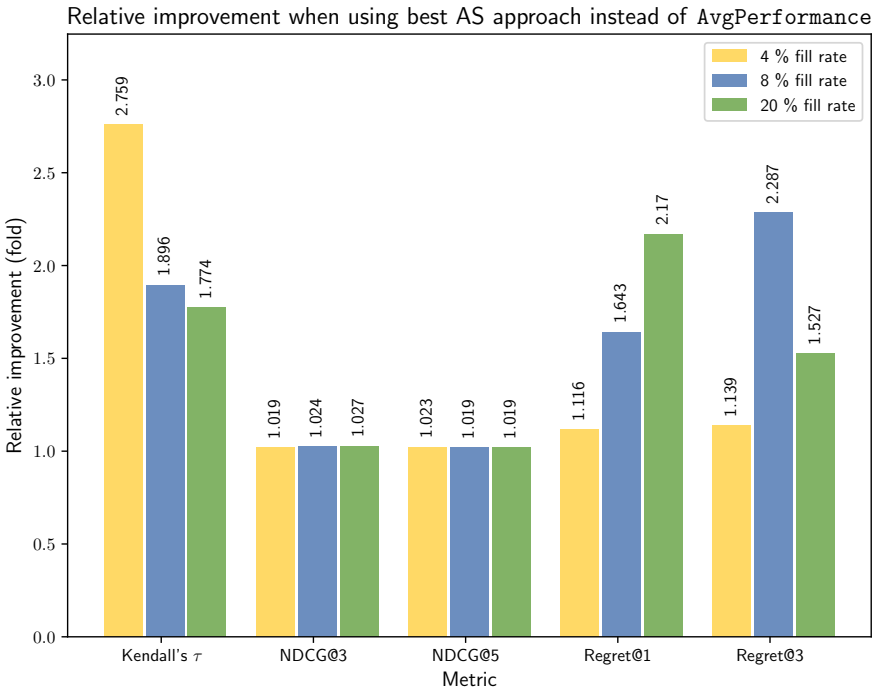

Fig. 3.7: This figure shows the relative improvement of the best approach for various fill rates in terms of the corresponding metric from Table 3.4 over the `AvgPerformance` baseline.

3.4.5.2 Benchmark Heterogeneity

Moreover, our study demonstrates the heterogeneity of the benchmark scenario. As described by Lindauer and Hoos [LH12], a relevant measure for heterogeneity is the per-instance potential for improvement over a solution that is static across instances, i.e. what is often called the single best algorithm or solver (SBS). In this case study, the SBS is represented by the `AvgPerformance` baseline, which is always worse than the oracle by definition on the regret@k measures and also on the other measures as there is another approach reaching better performance in all cases as Table 3.4 shows.

In order to further support the claims made above, Figure 3.7 visualizes the relative improvement of the best approach over the `AvgPerformance` baseline for each fill rate and each measure. It becomes evident that for a very small fill rate of 4% the largest improvement over the `AvgPerformance` is possible in terms of the Kendall's τ measure and hence in terms of improving the overall ranking. Considering the other metrics, an improvement is possible as well, but not to a large extent. This assessment drastically changes when increasing the fill rate to 8% and 20%, where

Tab. 3.4: Results for the performance metrics Kendall's tau (τ), NDCG@k (N@3, N@5), and regret@k (R@1, R@3) for a varying number of performance value pairs used for training. The best performing approach is highlighted in bold, the second best is underlined, and significant improvements of the best approach over others are denoted by •.

Approach	4% fill rate / 25 performance value pairs				
	τ	N@3	N@5	R@1	R@3
PAReg	0.1712 •	0.9352 •	0.9433 •	0.0601 •	0.0185 •
Alors (NDCG)	0.0504 •	0.9205 •	0.9223 •	0.0686 •	0.0225
Alors (REGR)	0.0303 •	0.9117 •	0.9191 •	0.0794 •	0.0190 •
DR	0.3445	0.9523	0.9604	0.0381	0.0089
DFReg	**0.3819**	**0.9564**	**0.9652**	**0.0302**	**0.0079**
RandomRank	-0.0038 •	0.8933 •	0.9105 •	0.0878 •	0.0272 •
AvgPerformance	0.1384 •	0.9388 •	0.9433 •	0.0337	0.0090
1-NN LR	0.1227 •	0.9290 •	0.9310 •	0.0733 •	0.0230 •
2-NN LR	0.1303 •	0.9278 •	0.9310 •	0.0642 •	0.0193 •

Approach	8% fill rate / 50 performance value pairs				
	τ	N@3	N@5	R@1	R@3
PAReg	0.2537 •	0.9453	0.9594	0.0493	0.0136
Alors (NDCG)	0.0472 •	0.9155 •	0.9164 •	0.0614 •	0.0208
Alors (REGR)	0.0807 •	0.9172 •	0.9304 •	0.0754 •	0.0285 •
DR	**0.3950**	**0.9584**	**0.9685**	0.0322	**0.0087**
DFReg	0.3692	0.9573	0.9661	**0.0300**	0.0123
RandomRank	-0.0038 •	0.8933 •	0.9105 •	0.0878 •	0.0272 •
AvgPerformance	0.2083 •	0.9355 •	0.9508 •	0.0493 •	0.0199 •
1-NN LR	0.1059 •	0.9246 •	0.9296 •	0.0564 •	0.0209
2-NN LR	0.0874 •	0.9269 •	0.9343 •	0.0541 •	0.0206

Approach	20% fill rate / 125 performance value pairs				
	τ	N@3	N@5	R@1	R@3
PAReg	0.3003 •	0.9525	0.9632	0.0395	0.0107
Alors (NDCG)	0.0540 •	0.9220 •	0.9242 •	0.0542 •	0.0228 •
Alors (REGR)	0.1039 •	0.9160 •	0.9329 •	0.0604 •	0.0222 •
DR	**0.4507**	**0.9696**	0.9715	**0.0241**	**0.0055**
DFReg	0.4264	0.9629	**0.9720**	0.0292	0.0071
RandomRank	-0.0038 •	0.8933 •	0.9105 •	0.0878 •	0.0272 •
AvgPerformance	0.2541 •	0.9437 •	0.9536 •	0.0523 •	0.0084
1-NN LR	0.1152 •	0.9245 •	0.9318 •	0.0594 •	0.0249 •
2-NN LR	0.1142 •	0.9292 •	0.9350 •	0.0412	0.0176 •

the potential for improvement becomes slightly less in terms of Kendall's τ, but much larger for both regret versions.

Overall, this analysis demonstrates that one can improve over the AvgPerformance baselines using instance-specific algorithm selection on the proposed benchmark scenario.

3.5 Related Work

As mentioned earlier, there is very little work representing algorithms as features, in particular in the context of AS. The arguably most similar work was recently proposed by Pulatov and Kotthoff [PK20] and revised in [Pul+22], who also suggest leveraging a joint model exploiting both instance and algorithm features. In contrast to our algorithm feature representation in the form of algorithm hyperparameters, they suggest analyzing the source code of the algorithms and extracting features based on that. Starting with simple statistical code features such as the number of lines or the number of files in their first work [PK20], Pulatov et al. [Pul+22] additionally propose to compute features based on the abstract syntax tree computed from the source code in their follow up work. Similar to us, they show in an experimental study on a part of ASlib that leveraging algorithm features does indeed improve algorithm selection performance. Similarly, Hilario et al. [Hil+09] also suggest identifying characteristics of algorithms, which allow to generalize over algorithms, such as the optimized loss function, and summarize these in an ontology — a suggestion, which was later also mentioned by Vanschoren [Van10]. Hough and Williams [HW06] represent algorithms by their hyperparameters among other features and even go a step beyond and also represent the execution environment using features in the context of AS. More recently, de Nobel et al. [dWB21] describe a technique to represent CMA-ES variants based on timeseries describing their behavior during optimization.

Furthermore, as mentioned earlier, representing algorithm configurations by their hyperparameters is quite often done in the field of AC, in order to estimate the performance of a configuration, for example in SMAC [HHL; Lin+22] or GGA++ [Ans+15]. Similarly, in AutoML, in particular neural architecture search, complete pipelines or neural network architectures are represented in terms of features to estimate their performance. Representing a neural network architecture can, for example, be done using graph neural networks [KW16] which take a graph, i.e. the architecture itself, as input. For an overview of related approaches, we refer to White et al. [Whi+21].

Lastly, as detailed in Section 2.3.2.6, model-based CF approaches also inherently compute (latent) algorithm features. Apart from the literature mentioned earlier, famous examples can also be found in the field of AutoML [Yan+19; FSE18].

3.6 Conclusion and Future Work

In this chapter we discussed the extreme algorithm selection problem, featuring an extremely large amount of algorithms and sparse training data, and elaborated on the weaknesses of existing AS approaches in this setting. Moreover, we suggested to leverage algorithm features as part of a dyadic feature representation to generalize across algorithms and learn a single joint model across all algorithms as a way to cope with the challenges of the XAS problem. As part of this, we discussed the advantages and disadvantages of dyadic approaches and come to the conclusion that finding a suitable algorithm feature representation is a very challenging problem. To analyze the performance of our suggested approaches, we designed a heterogeneous benchmark scenario featuring more than one thousand algorithms. The results of our experimental evaluation on that benchmark scenario show that, for our particular case study, algorithm selectors with strong generalization performance can be obtained despite the small number of training examples using our dyadic feature approaches. Moreover, the results suggest that there is indeed a need for the development of specific methods addressing the characteristics of the XAS setting as standard approaches show a rather disappointing performance.

There are various paths to improve the work presented in this chapter of the thesis — we will focus on the three important ones. As should have become evident by now, designing good algorithm features, which go beyond the rather simple representation we chose, is one of these paths. One approach, which we deem particularly promising is the one of extracting discriminative algorithm features from timeseries describing the algorithm behavior during application de Nobel et al. [dWB21], which we also mentioned earlier. We believe that information about the behavior of the algorithm should be very useful in order to determine, if an algorithm will perform well on a new instance and should in principle also allow to generalize across algorithms behaving similarly. The second most important path to follow is the one of designing a good feature map ψ, which combines instance and algorithm features into a joint feature vector. While learning this map in an implicit fashion using a neural network works well in our approach, we believe that explicitly learning the map is a path that should be further explored. Lastly, in order to allow for easier research on the topic, more benchmark scenarios featuring both a large number of algorithms and algorithm features should be created and published in a unified format, for example as part of ASlib.

4 Offline Algorithm Selection Under Censored Feedback

In this chapter, we introduce the problem of so-called censored training data in algorithm selection (AS) and elaborate on why treating censored data is important in the context of AS and in how far existing methods struggle to do so (Section 4.1). To alleviate those problems, we suggest resorting to methods from the field of survival analysis, which we shortly introduce (Section 4.2). We then draw on survival analysis methods in order to design risk-averse algorithm selectors specifically tailored towards avoiding the selection of timeouting algorithms under loss functions that impose a high penalty on such timeouts (Section 4.3). In an extensive experimental evaluation on ASlib we show that existing approaches for treating censored data do indeed perform poorly and that our suggested approaches majorly improve over the state of the art in AS (Section 4.4). Before closing the chapter with a recap, we put our work into the context of related work (Section 4.5).

For the remainder of this chapter, we will assume the standard offline AS setting defined in Section 2.1.1. Moreover, we will assume that the considered AS loss function l is either the runtime of the selected algorithm or a penalized version thereof.

The content presented in this chapter of the thesis has been partly published in the form of a conference paper [Tor+20a].

4.1 The Problem of Censored Training Data

Recall that in the instance-specific offline AS setting, we assume to have training data given to the algorithm selectors in the form of evaluations of the loss function l, i.e. algorithm runtime, for the algorithms $\mathcal{A}$ on the training instances $\mathcal{I}_D$. However, in combinatorial optimization, some algorithms may take extremely long to solve some instances yielding so-called heavy-tailed runtime distributions [GSC97] (as

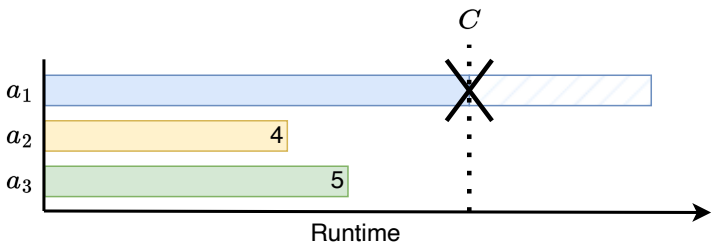

Fig. 4.1: This figure depicts how algorithms are often run in the context of AS. Here, algorithms a_2 and a_3 terminate before the cutoff C and thus feature a corresponding runtime. Algorithm a_1, however, is forcefully terminated as it did not finish until timestep C, and thus, C is only an upper bound on its runtime yielding a right-censored datapoint.

mentioned in Section 2.4). Due to this, the generation of training data for surrogate loss models in algorithm selection is usually time-constrained in the sense that a time limit C, called *cutoff*, is set when running an algorithm. If the algorithm does not solve the instance prior to this timeout, the execution is aborted to save computational costs. In such a case, the runtime data is *right-censored* [KK10] in the sense that $l(i, a) > C$, i.e. the true runtime is known to exceed C. This process is depicted in Figure 4.1. In cases where the data is not explicitly generated for training, but comes from a real-world scenario, it usually features only one true algorithm runtime for each instance, as in the absence of an algorithm selector, it is common practice to run many algorithms in parallel and stop all others as soon as the first one solves the instance. Hence, the rating/performance matrix $R \in \mathbb{R}^{|\mathcal{I}_D| \times |\mathcal{A}|}$ of known loss values spanned by the set of training instances $\mathcal{I}_D$ and algorithms $\mathcal{A}$, similar to the performance matrix depicted in Figure 2.7 in Chapter 2, is only partially filled with known runtimes. As an example, consider the algorithm selection benchmark ASlib [Bis+16], where in some cases, more than 70% of the data points are censored (cf. Table 2.5). Correspondingly, those datapoints should somehow be handled as they can make up a significant amount of the data to learn from.

4.1.1 Existing Solutions

A naïve approach to deal with this problem is to either impute missing runtimes with a default value or ignore them altogether when training a surrogate loss $\hat{l}$. Common choices for a default are the cutoff time C or ten times the cutoff time, motivated by the *penalized average runtime* (PAR10) score (cf. Section 2.4.1). As a short reminder,

the PAR10 score corresponds to the runtime, if the algorithm did not time out, or ten times the cutoff, if it timed out. However, all these strategies exhibit considerable drawbacks:

1. Any form of imputation is a deliberate distortion of the training data and thus should be done with care. In scenarios with many censored data points, e.g., the one with over 70%, imputation may lead to strongly biased surrogate models. More specifically, the imputation of missing values with the cutoff time, which is lower than the actual runtime, can lead to a systematic underestimation of true runtimes. In fact, one can even show theoretically that this is the case under certain assumptions as we elaborate on in more detail in Section 5.3.1.

2. Dropping censored samples altogether is a waste of valuable information. Although the censored samples do not inform about precise runtimes, they still carry information, namely that $l(i, a) > C$. Furthermore, by dropping the long and keeping the short runtimes, there is again a danger of inducing over-optimistic models.

Although we are not the first to remark these problems of censored data (cf. Section 4.5), very little work has been done on solving them in the context of algorithm selection. A method for imputing censored data points introduced by Schmee and Hahn [SH79] was studied in the context of algorithm configuration (AC) [HHL11; Egg+18] and AS [Xu+08]. A generalization of this method proposed by Hutter et al. [HHL11] and later decoupled from random forests by Eggensperger et al. [Egg+18], starts by fitting a model on the uncensored data points, and then uses it to predict the mean μ and the variance σ^2 of the distribution for each censored data point. Based on these statistics, a truncated normal distribution $\mathcal{N}(\mu, \sigma^2)_{\geqslant C}$, where C is a known lower bound on the true runtime, is computed. Lastly, each censored data point is imputed with the mean of its associated truncated normal distribution and the model is refit based on both the censored and uncensored data points. This process is repeated until a stopping criterion is reached. Note that this method relies on the assumption of a normal distribution for runtimes, which is in contradiction to the more common heavy-tail assumption [GSC97]. Accordingly, the runtime data to learn from is usually log-transformed before applying the method to comply with the heavy-tail assumption.

Despite the fact that this approach is a common way to deal with censored data in the context of AS and AC, the method was originally introduced for linear models

and shown to work well for problems with only a single feature, but there is no strong justification for why the method should work well for higher-dimensional problems and more powerful models. In practice, we observe that the method often fails to improve over training with the censored data directly in our experiments (see Figure 4.3).

4.2 Survival Analysis and Random Survival Forests

In this section, we explain some basic concepts of survival analysis (SA) required to understand the remainder of this thesis chapter. Moreover, we present an existing non-parametric approach to fitting survival distributions, based on tree induction and ensembling techniques, on the basis of which we construct an algorithm selector later in this chapter. For an in-depth introduction to SA, we refer to Kleinbaum and Klein [KK10].

4.2.1 Basic Concepts of Survival Analysis

In SA, we typically proceed from historical data of the form

$$\mathcal{D} = \{(\boldsymbol{x}_n, y_n, \delta_n)\}_{n=1}^{N} \,, \tag{4.1}$$

where $\boldsymbol{x}_n \in \mathcal{X} \subseteq \mathbb{R}^d$ is a d-dimensional feature representation of a context, $y_n \in \mathbb{R}_+$ is the observed time until the event of interest occurred for the given context, and $\delta_n \in \{0, 1\}$ indicates whether the sample is (right-)censored ($\delta_n = 1$) or uncensored ($\delta_n = 0$), i.e. y_n is the true time until the event occurred or a right-censored version thereof. More precisely,

$$y_n = \begin{cases} T_n & \text{if } \delta_n = 0 \\ C_n & \text{if } \delta_n = 1 \end{cases} \,, \tag{4.2}$$

where T_n is the uncensored and C_n the censored survival time one of instance n, which means that y_n is only a latent representation of T_n. Observe that the censored survival time C_n can be different for each instance in principle. Given such historical recordings, one is interested in inferring a model

$$z : \mathcal{X} \to \mathbb{R}_+ \,, \tag{4.3}$$

which, given a context $\boldsymbol{x}$, correctly predicts the unknown survival time $T \in \mathbb{R}_+$.

In our setting of algorithm selection, the event of interest is the termination of an algorithm, and a context is given by a problem instance, or more precisely, its feature vector. Moreover, since we consider a fixed cutoff C for a timeout for all instances and algorithms, $C_n = C$ for all $1 \leqslant n \leqslant N$.

Obviously, the problem of inducing Equation 4.3, as stated above, is very similar to the problem of standard regression. However, since the target variable is censored, the training data contains partial information and hence uncertainty. Accordingly, to capture this uncertainty, the majority of survival analysis approaches employ probabilistic models. To this end, the time until an event of interest T occurs is considered a random variable, which is modeled via a probability distribution using the so-called *survival function* (SF):

$$S(t, \boldsymbol{x}) = \mathbb{P}\left(T \geqslant t \mid \boldsymbol{x}\right) , \tag{4.4}$$

i.e. the probability that the event of interest occurs at time t or later, given context $\boldsymbol{x} \in \mathbb{R}^d$. Note that the survival function can be equivalently expressed in terms of the cumulative (death) distribution function (CDF) $F(t, \boldsymbol{x}) = \mathbb{P}\left(T \leqslant t \mid \boldsymbol{x}\right)$ as $S(t, \boldsymbol{x}) = 1 - F(t, \boldsymbol{x})$. While the survival function is defined according to the non-occurrence of the event until a certain time, the *hazard function*

$$h(t, \boldsymbol{x}) = \lim_{\Delta_t \to 0} \frac{\mathbb{P}\left(t \leqslant T < t + \Delta_t \mid T \geqslant t, \boldsymbol{x}\right)}{\Delta_t} \tag{4.5}$$

can be interpreted as expressing a degree of propensity of the event to occur at time t, under the condition that it did not occur before. The above functions are closely related to each other, and knowing one of them suffices to derive the others:

$$S(t, \boldsymbol{x}) = \exp\left(- \underbrace{\int_0^t h(u, \boldsymbol{x}) du}_{H(t, \boldsymbol{x})} \right) , \qquad h(t, \boldsymbol{x}) = - \left(\frac{dS(t, \boldsymbol{x})/dt}{S(t, \boldsymbol{x})} \right) \tag{4.6}$$

where $H(t, \boldsymbol{x})$ is the *cumulative hazard function*. To further understand this connection, consider the example of $S(t, \boldsymbol{x}) = \exp\left(-\lambda \cdot t\right)$ and $h(t, \boldsymbol{x}) = \lambda$, where the hazard rate is modeled as a constant λ independent of the context. In practice, this assumption will, of course, be overly simplistic, and the hazard function will depend on the context features $\boldsymbol{x}$ (also called *covariates* in SA). Moreover, to model the dependence on t, several parametric families of functions for instantiating the hazard function have been proposed, such as log-normal and Weibull functions.

A survival model can be used in various ways to obtain a real-valued prediction of the survival time, as requested by Equation 4.3. A natural predictor is the expected survival time, i.e. the expectation of the random variable T:

$$z(\boldsymbol{x}) := \mathbb{E}[T] = \int_0^\infty S(t, \boldsymbol{x})\, dt = \int_0^\infty 1 - F(t, \boldsymbol{x})\, dt \tag{4.7}$$

4.2.2 Random Survival Forests

Due to their excellent predictive power, random forests are often chosen to tackle standard regression problems and also serve as a strong baseline in AS for modeling a surrogate model $\widehat{l}(i, a)$. For this reason, we chose to model the runtime distribution of an algorithm a via the survival function S_a in the form of a *random survival forest* [Ish+08], an adaptation of standard random forests for survival analysis.

While similar to standard random forests, random survival forests differ in the way they (a) build the individual survival trees and (b) generate predictions from these individual trees. Similar to the CART algorithm [Bre+84], the individual survival trees are binary trees that are built via a recursive splitting approach. Splits are chosen to maximize survival difference between child nodes, i.e. by maximizing the difference in observed times y associated with these nodes. This process is continued until at least one of possibly multiple stopping criteria is reached, e.g., a node must contain at least d_0 uncensored samples. After such a survival tree has been built, it can be used to estimate the cumulative hazard function $H(t, \boldsymbol{x})$ for a query instance $\boldsymbol{x}$ in the following way: Starting at the root node, one recursively determines which subtree the query instance $\boldsymbol{x}$ belongs to, depending on the split criterion of the considered node until a leaf node $e(\boldsymbol{x})$ is reached. Let

- $t_1 < \ldots < t_{N(e(\boldsymbol{x}))}$ be the distinct times of events associated with leaf node $e(\boldsymbol{x})$,
- $d_{t,e(\boldsymbol{x})}$ be the number of samples in node $e(\boldsymbol{x})$ where the event occurred at time t, i.e. with $\delta = 0$ and $y = t$
- and $Y_{t,e(\boldsymbol{x})}$ the number of samples where the event has not occurred until time step t, i.e. with $\delta = 1$ or $y > t$.

Then, the cumulative hazard function $H(t, \boldsymbol{x})$ is estimated using the Nelson-Aalen estimator [Nel72; Aal78] as follows:

$$H(t, \boldsymbol{x}) = \sum_{t_i < \min\left(t_{N(e(\boldsymbol{x}))}, t\right)} \frac{d_{t_i, e(\boldsymbol{x})}}{Y_{t_i, e(\boldsymbol{x})}} \tag{4.8}$$

This can be seen as an empirical estimation of the hazard function for each event time t_i, and an accumulation of these estimates over time. Note that the cumulative hazard function fully defines the survival distribution for a single survival tree. To obtain the distribution based on the entire forest, one simply takes the mean over the cumulative hazard function estimates of the single trees.

4.3 Survival Analysis for Algorithm Selection

To tackle the problem of algorithm selection using SA, we learn one survival distribution, i.e. algorithm runtime distribution, for each algorithm a separately, using the above-mentioned random survival forests. To this end, we leverage the training data available by constructing algorithm-specific training datasets of the form shown in Equation 4.1 as

$$\mathcal{D}_a = \left\{ \left(\boldsymbol{f}_i, l(i,a), [\![l(i,a) = C]\!] \right) \mid i \in \mathcal{I}_D \right\}, \tag{4.9}$$

where we assume that an occurrence of an algorithm runtime $l(i,a)$ equal to the cutoff time C, i.e. $l(i,a) = C$, indicates a timeout of algorithm a on instance i and hence a censored sample. Based on this, we estimate the associated survival distribution S_a for each algorithm a, which is fully defined by the associated cumulative hazard function H_a. Recall that $\boldsymbol{f}_i$ is the instance feature vector associated with instance i.

In practice, the random survival forests estimate survival *step* functions $\widehat{S}_a$ for each algorithm a. Thus, we can approximate the integral of the survival function as follows:

$$\int_0^\infty S_a(t, \boldsymbol{f}_i)\, dt \approx \sum_{t_k, t_{k+1} \in \{0\} \cup \mathcal{T}_a \cup \{C\}} t_k \cdot \left[\widehat{S}_a(t_k, \boldsymbol{f}_i) - \widehat{S}_a(t_{k+1}, \boldsymbol{f}_i) \right], \tag{4.10}$$

where $\mathcal{T}_a$ denotes the set of event times, i.e. points of termination, observed for algorithm a. Likewise, other integrals related to the survival function, such as the expected runtime, can be approximated.

4.3.1 Decision-Theoretic Algorithm Selection

A survival distribution S_a can be seen as a generative model, which, in contrast to standard regression models, not only provides a point estimate of an algorithm's runtime but characterizes the runtime of algorithm a on problem instance i in terms of a complete probability distribution of a random variable $T_{a,i}$. This is a rich source of information, which can be used to realize algorithm selection in a more sophisticated manner by means of a decision-theoretic approach, adopting the principle of *expected utility maximization* [Sch82], or, equivalently, *expected loss minimization*. More specifically, this principle suggests a very natural definition of a loss function surrogate $\widehat{l}$ and thus an algorithm selector $s : \mathcal{I} \longrightarrow \mathbb{R}$ according to the framework introduced in Section 2.3 as

$$s(i) := \arg\min_{a \in \mathcal{A}} \widehat{l}(i, a)\,, \tag{4.11}$$

with

$$\widehat{l}(i, a) = \mathbb{E}\left[\mathscr{L}(T_{a,i})\right] \tag{4.12}$$

where $\mathscr{L} : \mathbb{R}_+ \to \mathbb{R}_+$ is a *decision-theoretic loss function* which maps runtimes to real-valued loss degrees.

At first sight, a natural decision criterion is the expected survival time (cf. Equation 4.7), i.e. the expected algorithm runtime, which was also suggested by Gagliolo and Schmidhuber [GS06]. This is obtained as a special case of Equation 4.11 with $\mathscr{L}(t) = t$:

$$\mathbb{E}\left[\mathscr{L}(T_{a,i})\right] = \mathbb{E}\left[T_{a,i}\right] = \int_0^\infty S_a(t, \boldsymbol{i})\, dt\,. \tag{4.13}$$

However, the expected algorithm runtime may appear sub-optimal in cases where the AS loss function l of interest substantially punishes algorithms running into a timeout. For example, the PAR10 score (cf. Section 2.4) assigns the runtime of an algorithm as its loss if it adheres to the timeout, but 10 times the cutoff if it times out. This is not accounted for by the expected runtime in Equation 4.13, which considers all regions of the survival function as *equally important* and tends to underestimate the *risk* of a timeout.

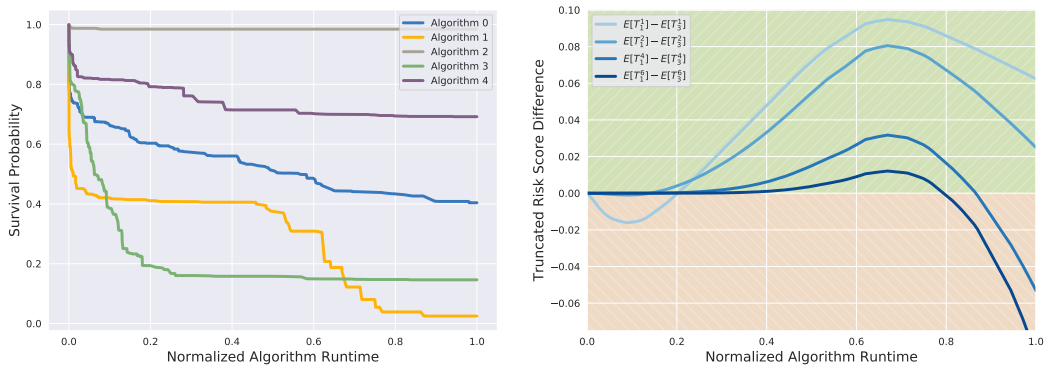

Fig. 4.2: Left: Survival functions of various algorithms on a problem instance. Right: Truncated risk score difference between the runtime of algorithm 1 (yellow on the left) and algorithm 3 (green), i.e. $\mathbb{E}\left[T_1^\alpha | T_1 \leqslant t\right] - \mathbb{E}\left[T_3^\alpha | T_3 \leqslant t\right]$. Values below the zero line for the non-truncated score indicate that $\mathbb{E}\left[T_1^\alpha\right] < \mathbb{E}\left[T_3^\alpha\right]$ and hence algorithm 1 is selected over algorithm 3, which is only the case with higher risk aversion. Furthermore, indeed larger values of α emphasize the tail of the distributions as the difference is close to zero for small values of t.

In fact, an algorithm can have a shorter runtime than another one *in expectation*, while having a higher probability of running into a timeout. To see this, consider the comparison between the first and third algorithm in Figure 4.2 on some example instance; time is normalized to make the cutoff $C = 1$. The left plot shows the survival functions of five algorithms computed based on Equation 4.6 and Equation 4.8. In the right plot, for now, we focus only on the light blue curve, which, for each point in time t, depicts how much longer algorithm 1 is *expected* to run compared to algorithm 3, given that both algorithms run for at most t, i.e. $\mathbb{E}\left[T_1 | T_1 \leqslant t\right] - \mathbb{E}\left[T_3 | T_3 \leqslant t\right]$. Looking at the value of this curve at $t = 1$, it is clear that algorithm 3 has a better expected runtime (Equation 4.13) than algorithm 1. However, the left plot also shows that algorithm 3 has a substantially higher probability to time out, as its probability to terminate at $t = 1$, i.e. at the cutoff, is larger than for algorithm 1. While a *risk-averse* view would focus on the survival probability at the timeout and then prefer algorithm 1 over algorithm 3, the expected algorithm runtime prefers 3 over 1, because it weights all parts of the distribution equally— clearly a sub-optimal choice when timeouts are strongly punished.

As an alternative, the PAR10 score itself may serve as a loss function:

$$\mathscr{L}(t) = \text{PAR10}(t) = \begin{cases} t & \text{if } t \leqslant C \\ 10 \cdot C & \text{else} \end{cases}. \tag{4.14}$$

The principle of expected loss minimization (Equation 4.11) then comes down to minimizing the *expected PAR10 score*, which was also suggested earlier by Biedenkapp et al. [Bie+17] as part of a study on algorithm parameter importance analysis. If this score plays the role of the *decision-theoretic loss function* and hence the loss function surrogate, on the basis of which the choice of an algorithm will eventually be assessed, this would indeed be a natural idea. On the other side, it is true that the PAR10 loss is rather extreme in the sense of strongly focusing on the probability mass close to the cutoff. While we argued that uniformly averaging over the entire range of runtimes (i.e. computing the expected runtime) is sub-optimal, this behavior could be overly conservative and hence not optimal either. To understand this, consider again Figure 4.2. The desire to avoid timeouts at any cost would require choosing algorithm 1. At the same time, however, a preference for algorithm 3 appears by no means absurd, since there is undeniably a considerable probability that this algorithm is substantially faster than algorithm 1 — all the more keeping in mind that the survival functions are only estimates of the underlying ground truth distributions, and hence $\mathbb{E}\left[\mathscr{L}(T_{a,i})\right]$ is only an estimate of the true expected loss.

4.3.2 Risk-Averse Algorithm Selection

In light of the above discussion, our general goal should be to trade off the different regions of the runtime distributions, keeping in mind the advantages of potentially very short runtimes on the one hand, and the risk of timeouts on the other hand. To this end, we consider two parameterized classes of functions for instantiating $\mathscr{L}$. Both functions are convex, thereby reflecting a *risk-averse* attitude in decision-making [Fis69]. In the context of algorithm selection, where decisions are algorithms and the avoidance of timeouts has the highest priority, risk aversion appears to be a very natural and meaningful property.

The first class of functions consists of polynomials

$$\mathscr{L}_\alpha(t) = t^\alpha\,, \tag{4.15}$$

where $\alpha \in \mathbb{R}_+$ controls the degree of risk aversion. Hence, larger values of $\alpha \geqslant 1$ result in stronger risk aversion and are therefore better suited for hard algorithmic problems with a large risk of timeouts, whereas smaller values $0 < \alpha < 1$ should be chosen for simpler problems where the majority of algorithms terminate before the cutoff time. One can observe that the higher α, the less important is the

behavior of the algorithms for low runtimes (the head of the distribution) and the more important for long runtimes (the tail of the distribution), leading to a clear preference of the "safer" alternatives for $\alpha > 2$. In our experimental evaluation, we demonstrate that α can be tuned effectively using hyperparameter optimization (HPO) techniques (cf. Section 2.2.4).

As a second class of functions, we consider functions of the form

$$\mathscr{L}_{\alpha,\beta}(t) = \min\{-\alpha \log(1-t), \beta\} \ , \tag{4.16}$$

where $\alpha \in [0,1]$ again defines the degree of risk aversion, with smaller values encouraging more risk-averse selections, i.e. potentially safer algorithm choices, and $\beta \in \mathbb{R}_+$ constitutes an upper bound on $\mathscr{L}$ to limit extreme behavior. Here, we assume that $t \in [0,1]$, which can be achieved via rescaling all runtimes such that $C = 1$. Again, both parameters α and β can be learned as we demonstrate in the experiments (cf. Section 4.4).

In practice, there is no obvious heuristical choice of a suitable function and corresponding parameters for a problem at hand. Therefore, we suggest to determine the most suitable function in a data-driven way using HPO techniques. In the experimental evaluation, we present an approach based on Bayesian optimization [Fra18], automatically selecting either the polynomial or the log-based function as well as adequate parameters α (and β) for a given scenario. This is similar to what is done by Lindauer et al. [Lin+15] in their tool called AutoFolio, who go a step beyond and automatically select the best AS approach from a set of AS approaches for a given scenario and automatically tune all of its hyperparameters.

4.4 Experimental Evaluation

In this section, we provide an extensive evaluation of the proposed methodology based on survival analysis, applying it to algorithm selection library (ASlib) (cf. Section 2.6 for an overview) and comparing it to state-of-the-art approaches. The list of the scenarios considered in this evaluation can be inferred from Figure 4.3.

4.4.1 Experimental Setup

We evaluate each approach by conducting a 10-fold cross-validation for every scenario where the concrete folds are provided by the corresponding scenario. Unsolvable instances, for which no valid solution could be determined by any algorithm before violating the fixed timeout, are subsequently removed from the test fold as all selectors are equally bad on them anyway.

Complete algorithm selection systems commonly integrate complementary techniques such as pre-solvers or feature selectors. For our experimental study, we consider pure algorithm selection models without any pre-solvers or feature selectors to focus on the models themselves and their suitability to deal with censored data. However, since some instance features are missing and for better comparability, we imputed missing feature values with their respective mean across all training instances for all considered approaches and baselines. Moreover, we standardize all features. The time for computing the corresponding instance feature representations required by the approaches is part of their runtime as well.

The performance of the approaches is assessed via the PAR10 metric (cf. Section 2.4.1) defined as the penalized average runtime of the selected solver required to solve problem instances. Here, timeout violations are explicitly penalized by a factor of 10. Recall that, whether the proposed techniques constitute improvements over the single best solver (SBS) baseline, and consequently close the gap to the virtual best solver (VBS), can be assessed in terms of the *normalized PAR10* (nPAR10) metric (cf. Section 2.4.1) defined as $\text{nPAR10} = {(PAR10_{Model} - PAR10_{VBS})}/{(PAR10_{SBS} - PAR10_{VBS})}$. Here, the SBS refers to the algorithm being best on average wrt. PAR10, whereas the (hypothetical) VBS denotes the optimal algorithm selector. Normalized scores greater than 1 denote that the approach is on average less efficient than the SBS, whereas scores below 1 indicate an improvement over the SBS baseline.

Our experimental study specifically distinguishes the examined baselines according to their performance under different treatments of censored data as discussed in Section 4.1. We initially examine which of the previously proposed methods to address censoring works best with each baseline, and subsequently compare their respective best results in terms of their PAR10 performance against our proposed methodology. Consequently, each baseline selector is evaluated under removal of censored data (*Ignored*), imputation by the cutoff (*Runtime*), and imputation by a

timeout penalty (*PAR10*). Where applicable, we also evaluate the method proposed by Schmee and Hahn [SH79] discussed in Section 4.1. We choose the most effective method for coping with censored data for each baseline selector based on the median PAR10 performance aggregated over all considered ASlib scenarios.

In the overall comparison, we include three variants of our approach leveraging random survival forests for estimating the survival functions. First, we consider a variant selecting algorithms according to their expected runtime as defined in Equation 4.13 (referred to as Run2SurviveExp). Second, a selection according to the expected PAR10 score of an algorithm as defined in Equation 4.14 (referred to as Run2SurvivePAR10) is evaluated. Lastly, a variant making use of the previously introduced polynomial and log-based functions is used, where (a) the type of the function and (b) respective parameters are tuned by means of Bayesian optimization on the training data (Run2SurvivePoly/Log).

The code for all experiments and the generation of plots, including detailed documentation, can be found on Github[1]. More technical details can be found in Section A.2.

4.4.2 Baselines

In the following, we give a short overview of all baselines we compare with. For a categorization of these approaches, refer to Section 2.3.

SUNNY [AGM14] retrieves the k nearest neighbors of a queried problem instance in terms of Euclidean distance w.r.t. the feature representation. From this set, the most efficient algorithm in terms of the (possibly imputed) training runtimes is chosen.

ISAC [Kad+10] leverages a g-means clustering algorithm instead of a kNN approach to partition the feature space. Originally designed for algorithm configuration, we transfer ISAC to algorithm selection by borrowing SUNNY's decision strategy. More specifically, we first assign the new instance to the closest centroid and subsequently select algorithms performing best on the training instances associated with the respective cluster.

[1] `https://github.com/alexandertornede/algorithm_survival_analysis`

SATzilla'11 [Xu+11] leverages pairwise comparisons of algorithms. For each pair of algorithms, a random forest is trained to decide which one performs better for a given problem instance. An explicit cost-sensitive loss function weighs instances subject to the absolute difference in their imposed computational cost. Faced with a new instance, the algorithm that wins the highest number of pairwise comparisons is predicted.

PerAlgorithmRegressor learns one random forest per candidate algorithm to predict each algorithm's performance while facilitating algorithm selection according to their respective estimates.

MultiClassSelector follows a multi-class classification approach, where instances are first labeled with the most effective selections, and a random forest is subsequently employed as a classifier to establish suitable selections.

4.4.3 Results

Figure 4.3 summarizes the results for the examined baselines subject to different methods of handling censoring. We find that imputing censored algorithm runs with their timeout works best for the PerAlgorithmRegressor, SATzilla'11, and SUNNY, whereas the PAR10 imputation is more effective for ISAC. We further find an iterative adjustment of censored values according to the technique proposed by Schmee and Hahn [SH79] to not benefit the PerAlgorithmRegressor in terms of median performance. Since an instance's most efficient solver is preserved regardless of whether censored algorithm runs are imputed with their exact timeout, PAR10 penalty or even entirely ignored, we note that the MultiClassSelector's performance is insensitive to prior imputation methods and thus not included in Figure 4.3. While treating censored values as their exact timeout or PAR10 score evidently yields similar results, obviously, discarding censored algorithm runs is the least appealing method, as depending on the ASlib scenario, up to 73.5% of runs are ignored.

Figure 4.4 compares the methods based on survival analysis proposed in this paper against the respectively most effective results w.r.t. the imputation procedure for each previously discussed baseline. The best results for each scenario are printed in bold, and an overline indicates beating all baselines. Evidently, neither ISAC nor the MultiClassSelector establish competitive algorithm selection on the ASlib benchmark, as they are consistently outperformed across all scenarios. This ob-

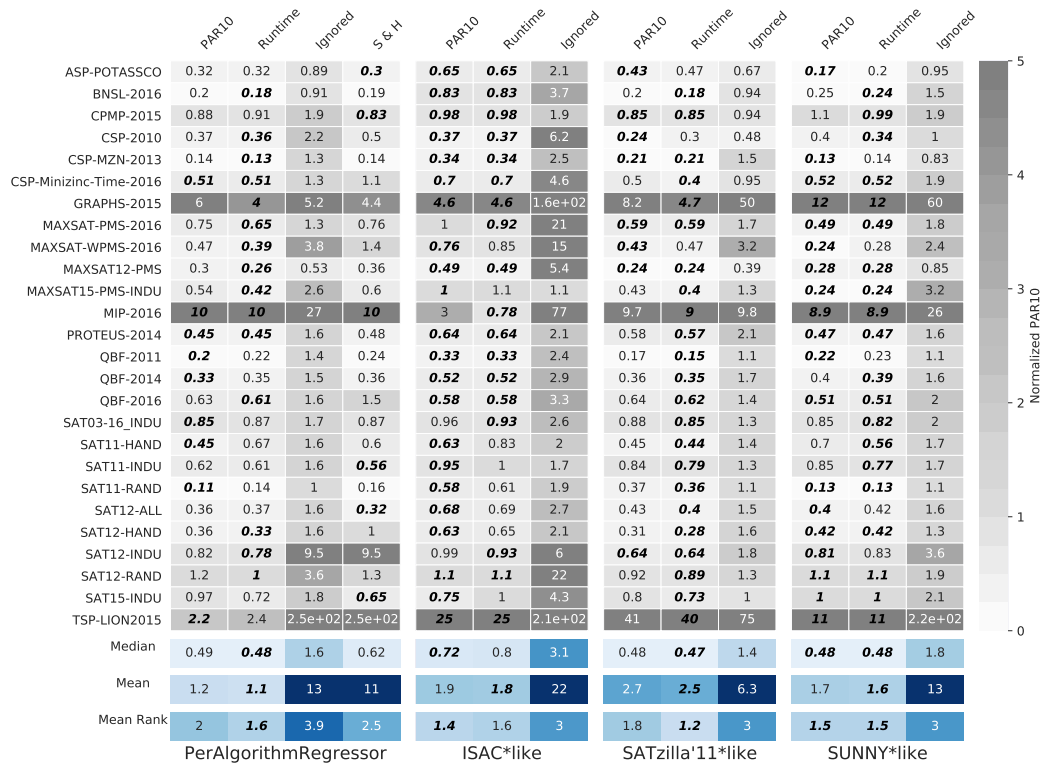

Fig. 4.3: Normalized PAR10 results of baselines for different ways of dealing with censored data: labeling data points as proposed by Schmee and Hahn [SH79] (S&H), with the PAR10 score (PAR10), the cutoff C (runtime), or the corresponding data points are ignored. The best results for each scenario are printed in bold.

servation is also reflected in the median/mean normalized PAR10 scores and the average ranks, which are much worse than the ones of all other approaches. The PerAlgorithmRegressor, SUNNY, and SATzilla'11, in contrast, broadly attain competitive results and consequently represent strong competitors to the random survival forests. However, Figure 4.4 illustrates each of our Run2Survive variants to outperform their adversaries in terms of median, mean nPAR10, as well as mean rank performance, aggregated across all scenarios. Compared to the baselines, Run2SurviveExp facilitates more effective per-instance algorithm selection on 8 scenarios, whereas Run2SurvivePAR10 yields superior results on 13 scenarios. While Run2SurvivePoly/Log beats all baselines in only 12 scenarios, it achieves an nPAR10 score below 1 on all scenarios except one, and hence consistently beats the SBS, which is not the case for any baseline. Furthermore, Run2SurvivePoly/Log beats the SBS in two scenarios where no other approach is able to.

Our findings further illustrate the usefulness of risk-averse decision-making in algorithm selection, as Run2SurviveExp can be improved upon in terms of median

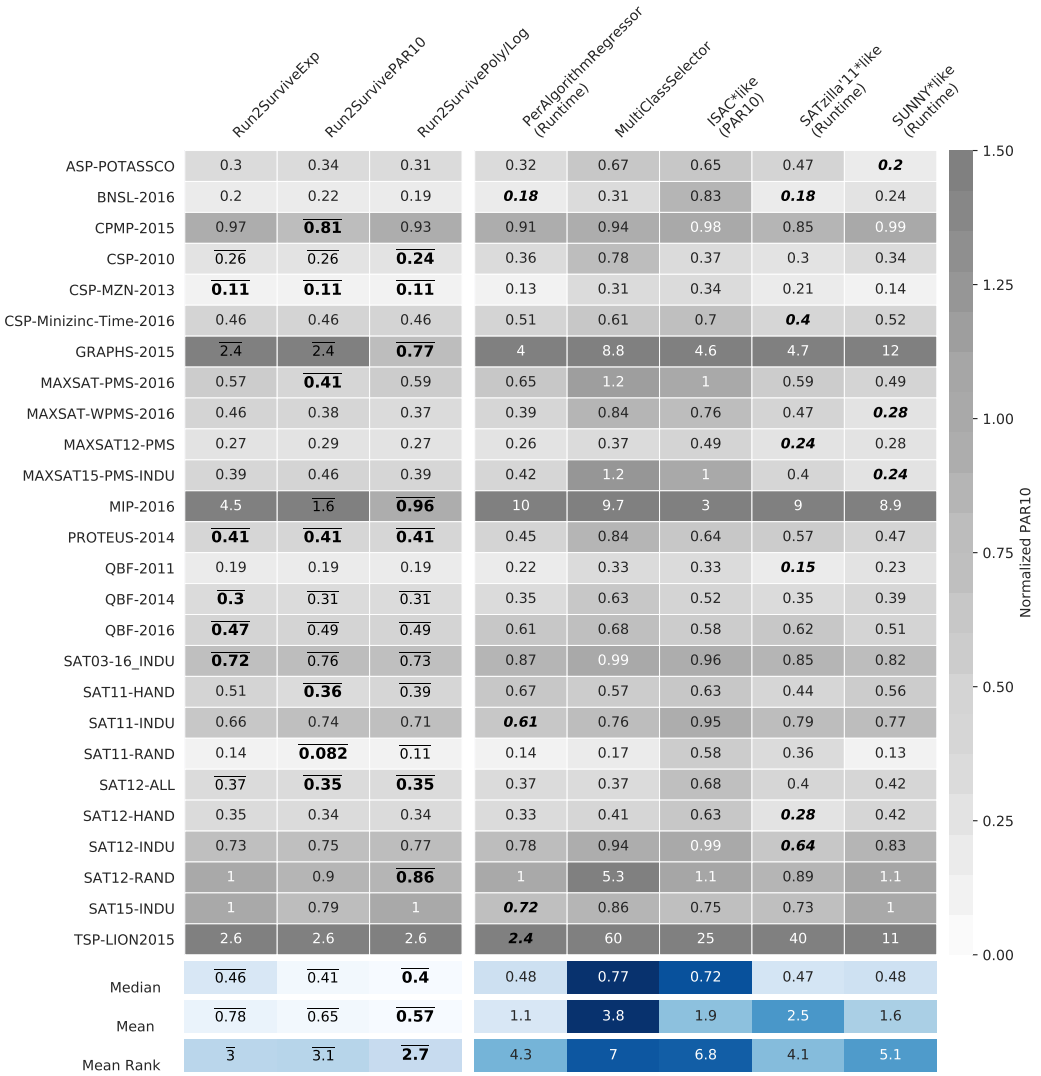

Fig. 4.4: Normalized PAR10 results where for each baseline the way of dealing with censored data is selected according to the minimum median across all examined scenarios. The best results for each scenario are printed in bold whereas an overline indicates beating all baselines.

and mean nPAR10 performance, and also in mean rank (Run2SurvivePoly/Log). In detail, Run2SurvivePAR10 and Run2SurvivePoly/Log identify superior selection criteria in 10 and 11 ASlib scenarios. However, they also degrade compared to Run2SurviveExp in 9 respectively 7 cases.

4.5 Related Work

To the best of our knowledge, the first and only work making direct use of survival analysis in the context of algorithm selection is by Gagliolo and Schmidhuber [GS06] and Gagliolo and Legrand [GL10]. Gagliolo and Schmidhuber [GS06] present an algorithm called GambleTA computing adaptive algorithm schedules for SAT and the Auction Winner Determination problem instances without any prior learning phase. GambleTA is a bandit-based algorithm deciding online which algorithms to run for how long such that it can use the obtained data to learn the runtime distributions of the considered algorithms. These distributions are modeled using the cumulative hazard function (CHF), which in turn are used to compute static schedules based on different techniques, such as selecting at timestep t the algorithm with e.g. minimal expected runtime. Using these static schedules, the authors also show how to construct dynamic schedules. In follow-up work, Gagliolo and Legrand [GL10] elaborate on the usefulness of survival analysis and potential pitfalls in the context of AS. Furthermore, they analyze statistical aspects of GambleTA.

Our work distinguishes itself from theirs mainly by the following points: First, Gagliolo and Schmidhuber [GS06] consider the setting of dynamic algorithm scheduling with a need for online learning, whereas we consider the standard setting of algorithm selection, whence other recommendations are required. Second, while the authors only consider two algorithmic problems and algorithm sets of size two, we consider the complete ASlib with over 25 scenarios from different problem domains featuring between 2 and 31 algorithms, making our experimental study much more extensive. Third, the authors consider nearest-neighbor motivated survival analysis models for modeling the CHF of the algorithms, while we make use of random survival forests due to their excellent empirical performance. Finally, and perhaps most importantly, we mainly focus on designing a risk-averse algorithm selection strategy, leveraging special decision-theoretic loss functions to better tailor selections towards the structure of loss functions similar to the PAR10 score.

Similar to our discussion in Section 4.1, Xu et al. [Xu+08] also remark the problems and implications of censored data in the context of algorithm runtime prediction. Instead of directly applying a method tailored towards such data, they apply the method by Schmee and Hahn [SH79] discussed in Section 4.1. Furthermore, similar to us, they perform a case study on how treating censored data during the training of runtime models impacts these models' performance. They find that (1) both ignoring and imputing censored data with the cutoff time does indeed yield overly

optimistic models (as we also note), and (2) that the method proposed by Schmee and Hahn [SH79] yields best performance in terms of the RMSE — a claim that we cannot verify.

Another interesting work featuring dealing with censored data in the context of AS was conducted by Hanselle et al. [Han+21][2], who suggest to treat such datapoints as interval-valued observations and to apply regression techniques based on the concept of superset learning [HC15]. However, they only slightly improve beyond imputing censored datapoints with the cutoff time, i.e. the strategy denoted with "Runtime" in Figure 4.3 such that we abstain from a direct empirical comparison.

In the context of AC, Hutter et al. [HHL11; Hut+14] also note problems arising from censored data when not treating these datapoints appropriately and describe a generalization of the method by Schmee and Hahn [SH79] to random forests making these better suited for censored data. Similarly, Biedenkapp et al. [Bie+17] and Eggensperger et al. [Egg+18] shortly mention the problem of censored data arising from terminating all running algorithm configurations when one out of many parallel runs finishes, and suggest to impute such censored data by previously discussed means. Eggensperger et al. [Egg+20] present a neural network based approach for AC, which specifically integrates censored information by optimizing the Tobit loss [Ame84].

4.6 Conclusion and Future Work

In this chapter, we proposed the use of survival analysis techniques combined with a decision-theoretic approach for the problem of algorithm selection. Taking advantage of the rich information provided by generative models of algorithm runtimes, together with the use of a risk-averse decision strategy to select the most promising algorithm for an unseen instance, we achieved a robust overall performance across different problem domains, such as SAT, CSP, and CPMP. Applying our method to a suite of 26 benchmark scenarios for algorithm selection from the standard benchmark library ASlib, we find it to be highly competitive and in many cases even superior to state-of-the-art methods. Moreover, considering several statistics across the considered benchmark datasets, our approach performs best in terms of average

[2]We note that we were involved in this work as well, but not as the main contributor.

rank as well as median/mean normalized PAR10 score across all scenarios, while achieving the best performances in up to 13 out of 26 scenarios.

There are various paths how to improve the ideas presented in this chapter in future work. Firstly, we deem it promising to find a means to learn a joint survival model across all algorithms as done in Chapter 3 instead of learning one model per algorithm as this would allow exploiting similarities between algorithms which almost certainly exist. Secondly, more work could be invested into testing other survival approaches such as survival SVMs [PNK15] in order to generate the algorithm runtime distributions on the basis of which the selection is performed. Lastly, one could investigate how Run2Survive behaves under less drastically penalized versions of the PARK, e.g. PAR3, and how the different decision-theoretic function classes we present have to be configured in such scenarios.

5 Online Algorithm Selection Under Censored Feedback

While in the last chapter we discussed the problem of censored data in the context of standard offline algorithm selection (AS) and how to tackle it using means of survival analysis (SA) and decision theory, this chapter considers the same problem in the online setting, i.e. online algorithm selection (OAS) (cf. Section 2.1.2). Recall that in the online setting, we assume that no training data is available and that instances arrive one by one in an iterative fashion raising the need for online learning.

Although the last chapter demonstrates that classical parameter-free survival analysis methods can perform very well in the offline AS setting, these methods cannot be easily transformed into online variants. For example, the well-known Cox proportional hazard model [Cox72] relies on estimating the baseline survival function through algorithms like the Breslow estimator (in its parameter-free version), which inherently requires the storage of all data in the form of so-called risk-sets [Bre72]. In principle, approximation techniques from the field of learning on data streams are conceivable [SH14]. Yet, in this chapter, we will focus on veritable online algorithms that do not require any approximation. To this end, we revisit the OAS problem from a bandit perspective (Section 5.1) and discuss related work from this new perspective (Section 5.6). Furthermore, we recap and detail how to model runtimes using linear models (Section 5.2). These models are then leveraged in Section 5.3, where we elaborate on how to use standard bandit algorithms to solve the OAS problem and on their disadvantages. Based on these observations, we propose theoretically grounded adaptations of those bandit approaches, which are built with the problem of censored data in mind (Section 5.4). Lastly, in our extensive experimental evaluation (Section 5.5), we find that these adapted approaches can be applied in order to successfully solve the OAS problem while featuring a time- and space-complexity independent of the time horizon.

Once again, in this chapter, we will assume that the loss function of interest is algorithm runtime or a penalized version thereof. Moreover, as already noted above, we assume the setting of OAS.

The content presented in this chapter of the thesis has been partly published in the form of a conference paper [TBH22].

5.1 The OAS Problem From a Bandit Perspective

Since this chapter will mostly feature algorithms from the multi-armed bandit literature, we will reformulate some important concepts and the OAS problem in the notion of the corresponding literature for better understandability and also as a means to see that the OAS problem can indeed be cast as a multi-armed bandits (MAB) problem. As a first step towards this, we reformulate the PARK score in the following.

5.1.1 Reformulation of the PARK

For ease of notation in this chapter, we reformulate the PARK (cf. Equation 2.39 in Section 2.4.1) as follows:

$$l(i,a) = m(i,a)[\![m(i,a) \leqslant C]\!] + \mathcal{P}(C)[\![m(i,a) > C]\!], \tag{5.1}$$

where $[\![\cdot]\!]$ is the indicator function [1] and $m : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}$ returns the true (stochastic) runtime of an algorithm a on an instance i. Moreover, $\mathcal{P} : \mathbb{R} \longrightarrow \mathbb{R}$ is a penalty function accounting for unsolved instances.

When selecting algorithm a, either the runtime $m(i,a)$ is observed, if $m(i,a) \leqslant C$, or a penalty $\mathcal{P}(C)$ due to a *right-censored* sample, i.e. $m(i,a) > C$, while the true runtime $m(i,a)$ remains unknown. With $\mathcal{P}(C) = 10C$, Equation 5.1 yields the well-known PAR10 score. Note that we particularly choose the PAR10 score as a loss function instead of runtime here and therefore denote runtime by m instead of l to avoid a clash of notation.

[1] It evaluates to 1, if the expression in the brackets evaluates to true and to 0 otherwise.

5.1.2 OAS as a Bandit Problem

OAS can be cast as a contextual MAB problem comprising a set of arms/algorithms $\mathcal{A}$ to choose from. In each round t, the learner is presented a context, i.e. an instance $i_t \in \mathcal{I}$ and its features $\boldsymbol{f}_{i_t} \in \mathbb{R}^d$, and is requested to select one of the algorithms for this instance, i.e. pull one of the arms, which will be denoted by a_t. As a result, the learner suffers a loss as defined in Equation 5.1.

In the stochastic variant of the contextual MAB problem, the losses are generated at random according to underlying distributions, one for each arm, which are unknown to the learner. Thus, the expected loss $\mathbb{E}\left[l(i_t, a_t)|\boldsymbol{f}_{i_t}\right]$ suffered at time step t is taken with respect to the unknown distribution of the chosen algorithm's runtime $m(i_t, a_t)$ and depends on the instance (features) $\boldsymbol{f}_{i_t}$. Ideally, in each time step t, the learner should pick one of the arms having the smallest expected loss for the current problem instance. Formally, the optimal strategy, called online oracle, is then defined as

$$s^*_{online}(h_t, i_t) = a^*_t \in \arg\min_{a \in \mathcal{A}} \mathbb{E}\left[l(i_t, a)|\boldsymbol{f}_{i_t}\right], \tag{5.2}$$

where, as the reader might recall, $h_t = \{(i_k, a_k, l_k)\}_{k=1}^{t-1} \in \mathcal{H}$ denotes the history of the selection process up to (but excluding) timestep t (cf. Section 2.1.2) [2]. In other words, the optimal strategy optimizes the expected PAR10. Needless to say, this optimal strategy can only serve as a benchmark, since the runtime distributions are unknown. Nevertheless, having an appropriate model or estimate for the expected losses, we can mimic the choice mechanism in Equation 5.2. To this end, we can learn a surrogate loss function $\widehat{l}_h : \mathcal{I} \times \mathcal{A} \longrightarrow \mathbb{R}$ giving rise to a suitable online algorithm selector of the following form:

$$s_{online}(h, i) := \arg\min_{a \in \mathcal{A}} \widehat{l}_h(i, a)\,. \tag{5.3}$$

Observe that in contrast to solutions to the offline AS problem, the surrogate loss function now depends on the history h. This is important as the surrogate loss function can (and has to be) updated online after each received feedback, considering that this is the main model powering the algorithm selection.

Due to the online nature of the problem, it is desirable that OAS approaches have a time- and space-complexity independent of the time horizon. In particular, memoriz-

[2] Note that this definition differs slightly from the one given in Section 2.1.2, as the expected value here explicitly depends on the instance features $\boldsymbol{f}_i$. While this is, in principle, always the case as long as the selector leverages such features, it is often not noted in practice. Here, however, we make this explicit as it will be important for the theoretical considerations formed in this chapter.

ing all instances (i.e. entire histories h_t) and constantly retraining in batch mode is no option. Moreover, from a practical point of view, decisions should be taken quickly to avoid stalling an AS system.

For convenience, we shall write from now on $l_{t,a}$ or $m_{i,a}$ instead of $l(i_t, a)$ or $m(i, a)$ for any $i, i_t \in \mathcal{I}$, $a \in \mathcal{A}$. Moreover, we write $\|\boldsymbol{x}\|_A := \sqrt{\boldsymbol{x}^\intercal A^{-1} \boldsymbol{x}}$ for any $\boldsymbol{x} \in \mathbb{R}^d$ and semi-positive definite matrix $A \in \mathbb{R}^{d \times d}$, and $\|\boldsymbol{x}\| := \sqrt{\boldsymbol{x}^\intercal \boldsymbol{x}}$.

5.2 Modeling Runtimes

As hinted at earlier, online algorithm selectors based on a bandit approach can be reasonably designed through the estimation of the expected losses occurring in Equation 5.2. To this end, we make the assumption that the logarithmic runtime of an algorithm $a \in \mathcal{A}$ on problem instance $i \in \mathcal{I}$ with features $\boldsymbol{f}_i \in \mathbb{R}^d$ depends linearly on those features up to some noise, i.e.

$$m_{i,a} = \exp(\boldsymbol{f}_i{}^\intercal \boldsymbol{\theta}_a^*) \exp(\epsilon_{i,a}), \tag{5.4}$$

where $\boldsymbol{\theta}_a^* \in \mathbb{R}^d$ is some unknown weight vector for each algorithm $a \in \mathcal{A}$, and $\epsilon_{i,a}$ is a stochastic noise variable with zero mean. The motivation for Equation 5.4 is twofold. First, theoretical properties such as positivity of the runtimes and heavy-tail properties of their distribution (by appropriate choice of the noise variables) are ensured. Second, we obtain a convenient linear model for the logarithmic runtime $y_{i,a}$ of an algorithm a on instance i, namely

$$y_{i,a} = \log(m_{i,a}) = \boldsymbol{f}_i{}^\intercal \boldsymbol{\theta}_a^* + \epsilon_{i,a}\,. \tag{5.5}$$

It is important to realize the two main implications coming with those assumptions. First, up to the stochastic noise term, the (logarithmic) runtime of an algorithm depends linearly on the features of the corresponding instance. This is not a big restriction, because the feature map $\boldsymbol{f}_i$ may include nonlinear transformations of "basic" features and play the role of a *linearization* [SS01] — we have demonstrated the usefulness of non-linear models in the last chapter. Moreover, previous work on AS has also considered logarithmic runtimes for model estimation [Xu+08]. Second, the model in Equation 5.5 suggests to estimate $\boldsymbol{\theta}_a^*$ separately for each algorithm, which is common practice but excludes the possibility to exploit (certainly existing) correlations between the performance of the algorithms. In principle, it might hence

be advantageous to estimate joint models as we have seen in Chapter 3. However, we abstain from doing so here in order to first develop a simple OAS approach analogously to the offline approaches.

Additionally, we assume that (1) $\exp(\boldsymbol{f}_i{}^\intercal \boldsymbol{\theta}_a^*) \leqslant C$ for any $a \in \mathcal{A}$, $i \in \mathcal{I}$ and (2) $\epsilon_{i,a}$ is normally distributed with zero mean and standard deviation $\sigma > 0$. While the first assumption is merely used for technical reasons, namely to derive theoretically valid confidence bounds for estimates of the weight vectors, the second assumption implies that $\exp(\epsilon_{i,a})$ is log-normal, which is a heavy-tail distribution yielding, in turn, a heavy-tail distribution for the complete runtime $y_{i,a}$. This strategy is consistent with practical observations that algorithm runtimes exhibit such heavy-tail distributions discussed earlier (cf. Section 4.1).

5.3 Stochastic Linear Bandits Approaches

As Equation 5.5 implies $\mathbb{E}\left[\log(m_{i,a}) | \boldsymbol{f}_i\right] = \boldsymbol{f}_i{}^\intercal \boldsymbol{\theta}_a^*$, it is tempting to apply a straightforward contextualized MAB learner designed for expected loss minimization, in which the expected losses are linear with respect to the context vector, viewing the logarithmic runtimes as the losses of the arms. This special case of contextualized MABs, also known as the *stochastic linear bandit* problem, has received much attention in the recent past [LS20]. Generally speaking, such a learner tends to choose an arm having a low expected log-runtime for the given context (,i.e. instance features), which in turn has a small expected loss. A prominent learner in stochastic linear bandits is LinUCB [Chu+11], a combination of online linear regression and the UCB [ACF02] algorithm. UCB implements the principle of optimism in the face of uncertainty and solves the exploration-exploitation trade-off by constructing confidence intervals around the estimated mean losses of each arm, and choosing the most optimistic arm according to the intervals.

Under the runtime assumption of Equation 5.5, the basic LinUCB variant (which we call BlindUCB) disregards censored observations in the OAS setting, and therefore considers the ridge regression (RR) estimator for each algorithm $a \in \mathcal{A}$ only on the non-censored observations. Formally, in each time step t, the RR estimate $\widehat{\boldsymbol{\theta}}_{t,a}$ for the weight parameter $\boldsymbol{\theta}_a^*$ is

$$\widehat{\boldsymbol{\theta}}_{t,a} = \underset{\boldsymbol{\theta} \in \mathbb{R}^d}{\arg\min} \sum_{\tau=1}^{t} [\![a_\tau = a, m_{i_\tau,a} \leqslant C]\!] \left(\boldsymbol{f}_{i_\tau}^\intercal \boldsymbol{\theta} - y_{i_\tau,a} \right)^2 + \lambda \|\boldsymbol{\theta}\|^2 , \tag{5.6}$$

where $\lambda \geqslant 0$ is a regularization parameter. The resulting selection strategy for choosing algorithm a_t at time t is then

$$a_t \in \arg\min_{a \in \mathcal{A}} \boldsymbol{f}_{i_t}^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot w_{t,a}(\boldsymbol{f}_{i_t}) \,, \tag{5.7}$$

where $\alpha > 0$ is some parameter controlling the exploration-exploitation trade-off, and

$$w_{t,a}(\boldsymbol{f}_{i_t}) = \|\boldsymbol{f}_{i_t}\|_{A_{t,a}} \tag{5.8}$$

the confidence width for the prediction of the (logarithmic) runtime of algorithm a for problem instance i_t based on the estimate in Equation 5.6.

Here, $A_{t,a} = \lambda I_d + X_{t,a}^\intercal X_{t,a}$ is the (regularized) Gram matrix, with I_d the $d \times d$ identity and $X_{t,a}$ denoting the design matrix at time step t associated with algorithm a, i.e. the matrix that stacks all the features row by row whenever a has been chosen.

The great appeal of this approach is the existence of a closed-form expression of the RR estimate, which can be updated sequentially with time- and space-complexity depending only on the feature dimension but independent of the time horizon [LS20] (cf. also Section A.4.3).

However, as already mentioned in Section 4.1, disregarding the right-censoring of the data often yields a rather poor performance of a regression-based learner in offline AS problems, so it might be advantageous to adapt this method to that end.

5.3.1 Imputation-Based Upper Confidence Bounds

A simple approach to include right-censored data into BlindUCB is to impute the corresponding samples by the cut-off time C as discussed in Section 4.1, giving rise to the RR estimate

$$\widehat{\boldsymbol{\theta}}_{t,a} = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \sum_{\tau=1}^{t} [\![a_\tau = a]\!] \left(\boldsymbol{f}_{i_\tau}^\intercal \boldsymbol{\theta} - \widetilde{y}_{i_\tau,a}\right)^2 + \lambda \|\boldsymbol{\theta}\|^2 \,, \tag{5.9}$$

where $\widetilde{y}_{i_\tau,a} = \min(y_{i_\tau,a}, \log(C))$ is the possibly imputed logarithmic runtime. Note that we impute with $\log(C)$ instead of C since we are modeling logarithmic runtimes.

When considering censoring, the least-squares formulation in Equation 5.9 has an important disadvantage. Those weight vectors resulting in an estimation of the runtime above C in case of a timeout are penalized (quadratically) for predictions $C < \widehat{y} < m(i,a)$, although these predictions are actually closer to the unknown ground truth than C. In fact, one can verify this intuition theoretically by showing that, for $\lambda = 0$, the RR estimate $\widehat{\theta}_{t,a}$ is downward biased in the case of censored samples as, for example, done by Greene [Gre05]. It is important to note that this bias is caused by a censoring of the runtimes, i.e. of the signal, and not by a truncation of the inputs or sparse (or non-representative) features as in [Dim+19].

Although the imputation strategy mentioned above has been shown to work astonishingly well in practice in offline AS [Tor+20a], the bias in the RR estimates requires a bias-correction in the confidence bounds of BlindUCB to ensure that the estimate falls indeed into the bounds with a certain probability. The corresponding bias-corrected confidence bound widths are

$$w_{t,a}^{(bc)}(\boldsymbol{f}_{i_t}) = \left(1 + 2\log(C)\left(1 + \sqrt{N_{a,t}^{(C)}}\right)\right) w_{t,a}(\boldsymbol{f}_{i_t}) \ , \tag{5.10}$$

where $N_{a,t}^{(C)}$ is the number of timeouts of algorithm a until t (cf. Section A.4.1 of the appendix for a derivation of the bias-corrected bounds). The resulting LinUCB variant, which we call BClinUCB (bias-corrected LinUCB), chooses the final algorithm in the same way as was shown in Equation 5.7, but uses $w_{t,a}^{(bc)}$ instead of $w_{t,a}$ and the runtime (RR) estimate in Equation 5.9.

Unfortunately, once again, these bias-corrected confidence bounds reveal a decisive disadvantage in practice, namely, the confidence bound of an algorithm $a \in \mathcal{A}$ is usually much larger than the actually estimated (log-)runtime $\boldsymbol{f}_{i_t}^\intercal \widehat{\boldsymbol{\theta}}_{t,a}$ for instance i_t. Therefore, the UCB value of a — let us call it $\delta_{t,a} = \boldsymbol{f}_{i_t}^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - w_{t,a}^{(bc)}(\boldsymbol{f}_{i_t})$ — is such that $\delta_{t,a}$ is smaller for algorithms with more timeouts due to the $N_{a,t}^{(C)}$ term. This prefers algorithms that experienced a timeout over those that did not. This, in turn, explains the poor performance of the BClinUCB strategies in the evaluation in Section 5.5.

5.3.2 Randomization of Upper Confidence Bounds

One way of mitigating the problem of the bias-corrected confidence bounds is to leverage a generalized form of UCB, called randomized UCB (RandUCB) [Vas+20], where the idea is to multiply the bias-corrected bounds $w_{t,a}^{(bc)}(\boldsymbol{f}_{i_t})$ with a random

realization of a specific distribution having positive support. RandUCB can be thought of as a mix of the classical UCB strategy, where the exploration-exploitation trade-off is tackled via the confidence bounds, and Thompson sampling [Tho33; Rus+18], which leverages randomization in a clever way for the same purpose (see next section). To this end, we define randomized confidence widths

$$\widetilde{w}_{t,a}(\boldsymbol{f}_{i_t}) = w_{t,a}^{(bc)}(\boldsymbol{f}_{i_t}) \cdot r \ , \tag{5.11}$$

where $r \in \mathbb{R}$ is sampled from a half-normal distribution with zero mean and standard deviation $\widetilde{\sigma}^2$. This ensures that $r \geqslant 0$ and that the confidence widths do indeed shrink when the distribution is properly parameterized. Although this improves the performance of LinUCB as we will see later, the improvement is not significant enough to achieve competitive results.

All variants of LinUCB for OAS introduced so far, i.e. BlindUCB, BClinUCB and RandUCB, can be jointly defined as in Algorithm 1. Note that theoretical details on all pseudocodes can be found in Section A.4.3.

5.3.3 Bayesian Approach: Thompson Sampling

As the confidence bounds used by LinUCB seem to be a problem in practice, one may think of Thompson sampling (TS) as an interesting alternative. The idea of TS is to assume a prior loss distribution for every arm, and in each time step, select an arm (i.e. algorithm) according to its probability of being optimal, i.e. according to its posterior loss distribution conditioned on all of the data seen so far. In particular, this strategy solves the exploration-exploitation trade-off through randomization driven by the posterior loss distribution.

More specifically, let the (multivariate) Gaussian distribution with mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$ be denoted by $\mathrm{N}\big(\boldsymbol{\mu}, \Sigma\big)$. Similarly, the cumulative distribution function of a (univariate) Gaussian distribution with mean $\mu \in \mathbb{R}$ and variance σ^2 at some point $z \in \mathbb{R}$ is denoted by $\Phi_{\mu,\sigma^2}(z)$. A popular instantiation of TS for stochastic linear bandits [AG13] assumes a Gaussian prior distribution $\mathrm{N}\big(\widehat{\boldsymbol{\theta}}_{t,a}, \sigma A_{t,a}^{-1}\big)$ for each weight vector of an algorithm a, where $\lambda, \sigma > 0$ and $\widehat{\boldsymbol{\theta}}_{t,a}$ denotes the RR estimate defined as in Equation 5.9. This yields $\mathrm{N}\big(\widehat{\boldsymbol{\theta}}_{t+1,a}, \sigma A_{t+1,a}^{-1}\big)$

Algorithm 1 `LinUCB` variants

Input parameters $\lambda \geqslant 0, \alpha, C > 0$ half-normal parameter $\widetilde{\sigma}$ for RandUCB
Initialization
for all $a \in \mathcal{A}$ **do**
  $A_{t,a} = \lambda I_{d\times d}, \boldsymbol{b}_{t,a} = 0_{d\times 1}, \widehat{\boldsymbol{\theta}}_{t,a} = 0_{d\times 1}, \widehat{l}_{t,a} = 0, N_{t,a}^{(C)} = 0$
end for
Main Algorithm
for time steps $t = 1 \ldots, T$ **do**
  Observe instance i_t and its features $\boldsymbol{x}_t = f(i_t) \in \mathbb{R}^d$
  if $t \leqslant |\mathcal{A}|$ **then**
    Take algorithm $a_t \in \mathcal{A}$ and obtain $y_t = \min(\log(m(i_t, a_t)), \log(C))$
  else
    for all $a \in \mathcal{A}$ **do**
      $\widehat{\boldsymbol{\theta}}_{t,a} \leftarrow (A_{t,a})^{-1}\boldsymbol{b}_{t,a}$
      $$\widehat{l}_{t,a} \leftarrow \begin{cases} \boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot w_{t,a}(\boldsymbol{x}_t), & \text{BlindUCB} \\ \boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot w_{t,a}^{(bc)}(\boldsymbol{x}_t), & \text{BClinUCB} \\ \boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot |r| \cdot w_{t,a}^{(bc)}(\boldsymbol{x}_t), \quad r \sim N(0, \widetilde{\sigma}^2) & \text{RandUCB} \end{cases}$$
    end for
    Take algorithm $a_t \in \arg\min_{a\in\mathcal{A}} \widehat{l}_{t,a}$ and
    obtain $y_t = \min(\log(m(i_t, a_t)), \log(C))$
  end if
  Updates:
  $N_{t,a_t}^{(C)} \leftarrow N_{t,a_t}^{(C)} + [\![m(i_t, a_t) > C]\!]$
  $$A_{t,a_t} \leftarrow \begin{cases} A_{t,a_t} + \boldsymbol{x}_t\boldsymbol{x}_t^\intercal, & \text{(BClinUCB,RandUCB)} \\ A_{t,a_t} + [\![m(i_t, a_t) \leqslant C]\!]\boldsymbol{x}_t\boldsymbol{x}_t^\intercal, & \text{(BlindUCB)} \end{cases}$$
  $$\boldsymbol{b}_{t,a_t} \leftarrow \begin{cases} \boldsymbol{b}_{t,a_t} + y_t\boldsymbol{x}_t, & \text{(BClinUCB,RandUCB)} \\ \boldsymbol{b}_{t,a_t} + [\![m(i_t, a_t) \leqslant C]\!]y_t\boldsymbol{x}_t, & \text{(BlindUCB)} \end{cases}$$
end for

The role of each hyperparameter is as follows:

- $\lambda > 0$ is a regularization parameter due to the considered RR.
- $\alpha > 0$ essentially controls the degree of exploration as a multiplicative term of the confidence width (see Equation 5.7). The higher (lower) it is chosen, the more (less) exploration is conducted.
- $C > 0$ is the cutoff time and depends on the considered problem scenario (specified by the algorithm selection library (ASlib) library).
- $\widetilde{\sigma} > 0$ is (only) used for RandUCB in order to specify the random sample's variance within the confidence width (see Sec. 5.3.2). The higher (lower) its choice, the larger (smaller) the effective exploration term, i.e. $|r| \cdot w_{t,a}^{(bc)}(\boldsymbol{x}_t)$.

as the posterior distribution at time step $t+1$. The selected algorithm is then defined by

$$a_t \in \arg\min_{a\in\mathcal{A}} \boldsymbol{f}_{i_t}^\intercal \widetilde{\boldsymbol{\theta}}_a \,, \tag{5.12}$$

where $\widetilde{\boldsymbol{\theta}}_a \sim \mathrm{N}\big(\widehat{\boldsymbol{\theta}}_{t,a}, \sigma A_{t,a}^{-1}\big)$ is sampled from the corresponding distribution for each $a \in \mathcal{A}$. Note that, as before, we assume to impute censored samples with the (log) cutoff time. Interestingly, as the experiments will show later on, this rather naïve version of Thompson sampling in the presence of censored data works astonishingly well in practice.

5.4 Expected PAR10 Loss Minimization

Due to the possibility of observing only a censored loss realization, i.e. $\mathcal{P}(C)$, it is reasonable to believe that a successful online algorithm selector needs to be able to properly incorporate the probability of observing such a realization into its selection mechanism. For this purpose, we derive the following decomposition of the expected loss under the assumptions made in Section 5.2:

$$\begin{aligned}\mathbb{E}\left[l_{t,a}|\boldsymbol{f}_{i_t}\right] &= \mathbb{E}\left[l_{t,a}|m_{i_t,a} \leqslant C, \boldsymbol{f}_{i_t}\right] \cdot \mathbb{P}(m_{i_t,a} \leqslant C|\boldsymbol{f}_{i_t}) \\ &\quad + \mathbb{E}\left[l_{t,a}|m_{i_t,a} > C, \boldsymbol{f}_{i_t}\right] \cdot \mathbb{P}(m_{i_t,a} > C|\boldsymbol{f}_{i_t}) \\ &= \left(1 - \Phi_{\boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^*, \sigma^2}(\log(C))\right) \cdot (\mathcal{P}(C) - E_C) + E_C\end{aligned} \tag{5.13}$$

where

$$E_C = E_C(\boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^*, \sigma) = \exp(\boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^* + \sigma^2/2) \cdot \frac{\Phi_{0,1}(C_{i_t,a}^{(1)})}{\Phi_{0,1}(C_{i_t,a}^{(2)})} \tag{5.14}$$

is the conditional expectation of a log-normal distribution with parameters $\boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^*$ and σ^2 under cutoff C and

$$\begin{aligned} C_{i_t,a}^{(1)} &= \frac{\left(\log(C) - \boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^* - \sigma^2\right)}{\sigma} \\ C_{i_t,a}^{(2)} &= C_{i_t,a}^{(1)} + \sigma \end{aligned} \quad . \tag{5.15}$$

As such, the decomposition suggests that there are two core elements driving the expected loss of an algorithm a conditioned on a problem instance i_t with features $\boldsymbol{f}_{i_t}$: its expected log-runtime $\boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^*$ and its probability of running into a timeout, i.e.

$$\mathbb{P}(m(i_t, a) > C|\boldsymbol{f}_{i_t}) = \left(1 - \Phi_{\boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^*, \sigma^2}(\log(C))\right) \quad . \tag{5.16}$$

In the interest of readability, we defer the actual derivation to Section A.4.2.

5.4.1 LinUCB Revisited

Having the refined expected loss representation in Equation 5.13, one could simply plug in the confidence bound estimates used by LinUCB for the log-runtime predictions to obtain an online algorithm selector following the optimism in the face of uncertainty principle, i.e. using an estimate of the target value to be minimized (here the expected loss in Equation 5.13), which underestimates the target value with high probability.

To this end,

$$o_{t,a} = \boldsymbol{f}_{i_t}^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot w_{t,a}(\boldsymbol{f}_{i_t}) \tag{5.17}$$

be the optimistic and

$$p_{t,a} = \boldsymbol{f}_{i_t}^\intercal \widehat{\boldsymbol{\theta}}_{t,a} + \alpha \cdot w_{t,a}(\boldsymbol{f}_{i_t}) \tag{5.18}$$

the pessimistic estimate used by LinUCB (or its variants), where $w_{t,a}(\boldsymbol{f}_{i_t})$ is the confidence width of the corresponding LinUCB variant conditioned on the current instance. With this, at time t we choose the algorithm as

$$a_t \in \arg\min_{a \in \mathcal{A}} \left(1 - \Phi_{p_{t,a},\sigma}(\log(C))\right) \cdot \left(\mathcal{P}(C) - \widehat{E}_C^{(1)}\right) + \widehat{E}_C^{(2)}, \tag{5.19}$$

where

$$\begin{aligned}
\widehat{E}_C^{(1)} &= \exp\left(p_{t,a} + \frac{\sigma^2}{2}\right) \cdot \frac{\Phi_{0,1}(\widehat{C}_{i_t,a}^{(o)})}{\Phi_{0,1}(\widehat{C}_{i_t,a}^{(p)} + \sigma)}, \\
\widehat{E}_C^{(2)} &= \exp\left(o_{t,a} + \frac{\sigma^2}{2}\right) \cdot \frac{\Phi_{0,1}(\widehat{C}_{i_t,a}^{(p)})}{\Phi_{0,1}(\widehat{C}_{i_t,a}^{(o)} + \sigma)}, \\
\widehat{C}_{i_t,a}^{(p)} &= \frac{(\log(C) - p_{t,a} - \sigma^2)}{\sigma}, \\
\widehat{C}_{i_t,a}^{(o)} &= \frac{(\log(C) - o_{t,a} - \sigma^2)}{\sigma}.
\end{aligned}$$

As $o_{t,a}$ underestimates and $p_{t,a}$ overestimates $\boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^*$ it is easy to see that the terms in Equation 5.19 are underestimating the corresponding terms occurring in Equation 5.13 with high probability, respectively. The corresponding pseudocode can be found in Algorithm 2. Once again, note that theoretical details on all pseudocodes can be found in Section A.4.3.

Algorithm 2 `LinUCB` variants based on Equation 5.13 (_rev) versions

1: *Input parameters* $\lambda \geqslant 0, \sigma > 0, \alpha > 0, C > 0, \mathcal{P} : \mathbb{R} \to \mathbb{R}$, half-normal parameter $\tilde{\sigma}$ for RandUCB
2: *Initialization*
3: **for** all $a \in \mathcal{A}$ **do**
4: $\quad A_{t,a} = \lambda I_{d\times d}, \boldsymbol{b}_{t,a} = 0_{d\times 1}, \widehat{\boldsymbol{\theta}}_{t,a} = 0_{d\times 1}, \widehat{l}_{t,a} = 0, N_{t,a}^{(C)} = 0$
5: **end for**
6: *Main Algorithm*
7: **for** time steps $t = 1 \ldots, T$ **do**
8: $\quad$ Observe instance i_t and its features $\boldsymbol{x}_t = f(i_t) \in \mathbb{R}^d$
9: $\quad$ **if** $t \leqslant |\mathcal{A}|$ **then**
10: $\quad\quad$ Take algorithm $a_t \in \mathcal{A}$ and obtain $y_t = \min(\log(m(i_t, a_t)), \log(C))$
11: $\quad$ **else**
12: $\quad\quad$ **for** all $a \in \mathcal{A}$ **do**
13: $\quad\quad\quad \widehat{\boldsymbol{\theta}}_{t,a} \leftarrow (A_{t,a})^{-1}\boldsymbol{b}_{t,a}$
14: $\quad\quad\quad r \sim \mathrm{N}(0, \tilde{\sigma}^2)$ (only for RandUCB)
15: $\quad\quad\quad p_{t,a} \leftarrow \begin{cases} \boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} + \alpha \cdot w_{t,a}(\boldsymbol{x}_t), & \text{BlindUCB} \\ \boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} + \alpha \cdot w_{t,a}^{(bc)}(\boldsymbol{x}_t), & \text{BClinUCB} \\ \boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} + \alpha \cdot |r| \cdot w_{t,a}^{(bc)}(\boldsymbol{x}_t), & \text{RandUCB} \end{cases}$
16: $\quad\quad\quad o_{t,a} \leftarrow \begin{cases} \boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot w_{t,a}(\boldsymbol{x}_t), & \text{BlindUCB} \\ \boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot w_{t,a}^{(bc)}(\boldsymbol{x}_t), & \text{BClinUCB} \\ \boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \alpha \cdot |r| \cdot w_{t,a}^{(bc)}(\boldsymbol{x}_t), & \text{RandUCB} \end{cases}$
17: $\quad\quad\quad \widehat{C}_{i_t,a}^{(1,p)} \leftarrow \frac{\log(C) - p_{t,a} - \sigma^2}{\sigma}$
18: $\quad\quad\quad \widehat{C}_{i_t,a}^{(1,o)} \leftarrow \frac{\log(C) - o_{t,a} - \sigma^2}{\sigma}$
19: $\quad\quad\quad \widehat{C}_{i_t,a}^{(2,o)} \leftarrow \frac{\log(C) - o_{t,a}}{\sigma}$
20: $\quad\quad\quad \widehat{C}_{i_t,a}^{(2,p)} \leftarrow \frac{\log(C) - p_{t,a}}{\sigma}$
21: $\quad\quad\quad \widehat{l}_{t,a} \leftarrow \exp(o_{t,a} + \sigma^2/2) \cdot \frac{\Phi_{0,1}(\widehat{C}_{i_t,a}^{(1,p)})}{\Phi_{0,1}(\widehat{C}_{i_t,a}^{(2,o)})} + \big(1 - \Phi_{p_{t,a},\sigma}(\log(C))\big) \cdot \left(\mathcal{P}(C) - \exp(p_{t,a} + \sigma^2/2) \cdot \frac{\Phi_{0,1}(\widehat{C}_{i_t,a}^{(1,o)})}{\Phi_{0,1}(\widehat{C}_{i_t,a}^{(2,p)})}\right)$
22: $\quad\quad$ **end for**
23: $\quad\quad$ Take algorithm $a_t \in \arg\min_{a\in\mathcal{A}} \widehat{l}_{t,a}$
24: $\quad\quad$ and obtain $y_t = \min(\log(m(i_t, a_t)), \log(C))$
25: $\quad$ **end if**
26: $\quad$ *Updates:* Same as lines 19–21 of Alg. 1
27: **end for**

Compared to Algorithm 1 there are two additional parameters:

- $\sigma > 0$ which ideally should correspond to the standard deviation of the noise variables in the model assumption defined by Equation 5.4.
- $\mathcal{P} : \mathbb{R} \to \mathbb{R}$ the penalty function for penalizing unsolved problem instances (we used $\mathcal{P}(z) = 10z$ corresponding to the PAR10 score).

As our experiments will reveal later on, the issues of the LinUCB-based algorithms caused by the wide confidence bands are alleviated to a certain degree by incorporating the explicit PAR10 loss minimization. The ones caused by the biased RR estimate remain, however.

5.4.2 Thompson Sampling Revisited

Fortunately, the refined expected loss representation in Equation 5.13 can be exploited quite elegantly by Thompson Sampling using Gaussian priors as in Section 5.3.3. Our suggested instantiation of TS chooses algorithm $a_t \in \mathcal{A}$ according to

$$a_t \in \arg\min_{a \in \mathcal{A}} \left(1 - \Phi_{\boldsymbol{f}_{i_t}^\intercal \widetilde{\boldsymbol{\theta}}_a, \widetilde{\sigma}_{t,a}^2}(\log(C))\right)\left(\mathcal{P}(C) - \widetilde{E}_C\right) + \widetilde{E}_C, \tag{5.20}$$

where $\widetilde{\boldsymbol{\theta}}_a$ is a random sample from the posterior $\mathrm{N}\left(\widehat{\boldsymbol{\theta}}_{t,a}, \sigma A_{t,a}^{-1}\right)$, and the conditional expectation of a normal distribution parameterized based on this sample is $\widetilde{E}_C = E_C(\boldsymbol{f}_{i_t}^\intercal \widetilde{\boldsymbol{\theta}}_a, \widetilde{\sigma}_{t,a})$ with variance $\widetilde{\sigma}_{t,a}^2 = \sigma \|\boldsymbol{f}_{i_t}\|_{A_{t,a}}^2$. Algorithm 3 provides the pseudocode for this revisited Thompson algorithm and a variant inspired by the Buckley-James estimate we discuss in the following.

Although the TS approach just presented does involve consideration of the timeout probability, it still suffers from the problem that the estimates for $\boldsymbol{\theta}_a^*$ are downward-biased as they are based on the RR estimate obtained from imputing censored samples with the cutoff time C. In the spirit of the Kaplan-Meier estimator [KM58] from the field of survival analysis, Buckley and James [BJ79] suggested augmenting censored samples by their expected value according to the current model and then solving the standard least-squares problem (for an overview of alternative approaches, we refer to Miller and Halpern [MH82]). This idea is particularly appealing, as it allows for easy integration into online algorithms, due to its use of the least-squares estimator. Also, it has the potential to produce more accurate (i.e. less biased) estimates for $\boldsymbol{\theta}_a^*$. The integration of the augmentation suggested by Buckley and James [BJ79] is shown in lines 17–20 in Algorithm 3.

Algorithm 3 `(bj_)Thompson_rev`

Input parameters $\sigma > 0, \lambda \geqslant 0, \mathcal{P} : \mathbb{R} \to \mathbb{R}, C$, BJ $\in \{\text{TRUE}, \text{FALSE}\}$
for all $a \in \mathcal{A}$ **do**
  $A_{t,a} = \lambda \cdot I_{d \times d}, \boldsymbol{b}_{t,a} = 0_{d \times 1}, \widehat{\boldsymbol{\theta}}_{t,a} = 0_{d \times 1}, \widetilde{\sigma}^2_{t,a} = 0$ and $\widehat{l}_{t,a} = 0$
end for
for time steps $t = 1 \ldots, T$ **do**
  Observe instance i_t and its features $\boldsymbol{x}_t = f(i_t) \in \mathbb{R}^d$
  if $t \leqslant |\mathcal{A}|$ **then**
    Take algorithm $a_t \in \mathcal{A}$ and obtain $y_t = \min(\log(m(i_t, a_t)), \log(C))$
  else
    for all $a \in \mathcal{A}$ **do**
      $\widehat{\boldsymbol{\theta}}_{t,a} \leftarrow (A_{t,a})^{-1} \boldsymbol{b}_{t,a}$
      $\widetilde{\sigma}^2_{t,a} \leftarrow \sigma \|\boldsymbol{x}_t\|^2_{A_{t,a}}$
      Sample $\widetilde{\boldsymbol{\theta}}_a \sim \mathrm{N}\big(\widehat{\boldsymbol{\theta}}_{t,a}, \sigma(A_{t,a})^{-1}\big)$
      $\widehat{l}_{t,a} \leftarrow \big(1 - \Phi_{\boldsymbol{x}_t^\intercal \widetilde{\boldsymbol{\theta}}_a, \widetilde{\sigma}^2_{t,a}}(\log(C))\big) \cdot \big(\mathcal{P}(C) - E_C(\boldsymbol{x}_t^\intercal \widetilde{\boldsymbol{\theta}}_a, \widetilde{\sigma}_{t,a})\big) + E_C(\boldsymbol{x}_t^\intercal \widetilde{\boldsymbol{\theta}}_a, \widetilde{\sigma}_{t,a})$ (RHS of Equation 5.20)
    end for
    Take algorithm $a_t \in \arg\min_{a \in \mathcal{A}} \widehat{l}_{t,a}$
    and obtain $y_t = \min(\log(m(i_t, a_t)), \log(C))$
  end if
  if $y_t = \log(C)$ and BJ $= \text{TRUE}$ **then**
    Sample $\breve{\boldsymbol{\theta}}_{a_t} \sim \mathrm{N}\big(\widehat{\boldsymbol{\theta}}_{t,a_t}, \sigma(A_{t,a_t})^{-1}\big)$ (if $\exp(\boldsymbol{x}_t^\intercal \breve{\boldsymbol{\theta}}_{a_t}) \leqslant C$ sample again)
    $y_t \leftarrow \log(\boldsymbol{x}_t^\intercal \breve{\boldsymbol{\theta}}_{a_t})$
  end if
  $A_{t,a_t} \leftarrow A_{t,a_t} + \boldsymbol{x}_t \boldsymbol{x}_t^\intercal \quad \boldsymbol{b}_{t,a_t} \leftarrow \boldsymbol{b}_{t,a_t} + y_t \boldsymbol{x}_t$
end for

The role of each hyperparameter is as follows:

- the role of $\lambda, \mathcal{P}, C$ is the same as in Algorithm 2, respectively.
- $\sigma > 0$ specifies the magnitude of the posterior distribution's variance and is therefore slightly different to σ in Algorithm 2.
- BJ specifies whether the Buckley-James inspired imputation strategy described at the end of Sec. 5.4.2 (BJ $= \text{TRUE}$) or the näive imputation strategy (BJ $= \text{FALSE}$) should be deployed.

5.5 Evaluation

As usual, we base our evaluation on the standard algorithm selection benchmark library ASlib (cf. Section 2.6) and compare to the most relevant competitor approaches. The concrete list of scenarios used can be inferred from Table 5.1. Since ASlib was originally designed for offline AS, we do not use the train/test splits provided by the benchmark, but rather pass each instance one by one to the corresponding online approaches, ask them to select an algorithm and return the corresponding feedback. To increase evaluation robustness, we randomly shuffle the

Tab. 5.1: Average PAR10 scores and standard deviation of Thompson sampling variants and Degroote, where the best value for each scenario is printed in bold and the second best is underlined.

Approach	bj_thompson		thompson_rev		degroote_ϵ-greedy_LR	
Scenario						
ASP-POTASSCO	949.38	± 62.38	**902.64**	± 78.43	1047.13	± 46.50
BNSL-2016	9638.04	± 378.05	**9467.01**	± 252.52	12510.26	± 1291.03
CPMP-2015	8241.01	± 1164.85	8158.72	± 1268.83	**6991.97**	± 501.36
CSP-2010	8295.76	± 699.43	7892.67	± 692.83	**7593.13**	± 208.94
CSP-MZN-2013	8207.06	± 532.70	8171.21	± 594.49	**8034.62**	± 113.78
CSP-Minizinc-Time-2016	4811.54	± 409.79	**4759.50**	± 306.03	5258.70	± 406.91
GRAPHS-2015	4.1e+07	± 4.4e+06	4.2e+07	± 3.4e+06	**3.5e+07**	± 1.4e+06
MAXSAT-PMS-2016	2853.44	± 210.21	**2808.51**	± 218.55	3279.54	± 133.00
MAXSAT-WPMS-2016	6304.15	± 166.98	6592.87	± 210.25	**6287.21**	± 541.69
MAXSAT12-PMS	5347.39	± 291.87	5408.40	± 482.42	**5308.11**	± 129.30
MAXSAT15-PMS-INDU	3046.05	± 128.34	**3032.08**	± 90.71	3867.70	± 255.98
MIP-2016	**8081.57**	± 845.74	8746.73	± 1159.36	10644.68	± 3405.18
PROTEUS-2014	**13484.34**	± 541.83	14115.69	± 768.16	15622.29	± 784.60
QBF-2011	15708.25	± 784.81	15178.86	± 904.72	**13912.24**	± 356.69
QBF-2014	**3629.40**	± 220.68	3679.96	± 256.03	4116.15	± 116.27
QBF-2016	5082.59	± 718.71	**5045.16**	± 848.59	5346.29	± 210.05
SAT03-16_INDU	**11980.15**	± 193.67	12154.46	± 221.01	12754.50	± 200.55
SAT11-HAND	30484.08	± 1379.35	30085.51	± 764.32	**29544.70**	± 952.78
SAT11-INDU	17540.58	± 530.82	17028.84	± 479.15	**17018.24**	± 647.90
SAT11-RAND	**18061.78**	± 2770.70	19061.88	± 2522.11	21008.77	± 530.22
SAT12-ALL	**4720.22**	± 432.14	5132.48	± 395.74	5650.32	± 214.36
SAT12-HAND	**7443.01**	± 180.51	7509.02	± 199.39	7634.24	± 267.89
SAT12-INDU	**4511.68**	± 76.33	4945.79	± 228.37	4755.52	± 206.95
SAT12-RAND	**4008.79**	± 206.59	4523.33	± 170.56	5023.73	± 174.68
SAT15-INDU	**7700.27**	± 310.65	7856.08	± 522.84	8220.22	± 525.13
SAT18-EXP	25201.41	± 681.42	**24906.56**	± 540.36	25272.35	± 881.19
TSP-LION2015	**1226.11**	± 309.42	1411.06	± 329.16	1634.79	± 112.29
avgrank	**1.814815**		1.888889		2.296296	

instances of each scenario, repeat the evaluation ten times with different seeds, and always report average or median aggregations across those ten repetitions. As ASlib contains missing feature values for some instances in some scenarios, we imputed these using the mean feature value of all instances seen until that point. Moreover, features were scaled to unit vectors by dividing by their L_2 norm. If the according variant does not self-impute censored values, these were imputed with the cutoff time.

All code, including detailed documentation, can be found on GitHub[3]. More details regarding the experiments, including corresponding hyperparameter settings, can be found in Section A.3.

[3] `https://github.com/alexandertornede/online_as`

Instead of directly reporting PAR10 scores, we sometimes resort to reporting a normalized version called rePAR10 (relative PAR10), which is comparable across scenarios and defined with respect to the oracle. Although in standard AS one usually uses the nPAR10 (cf. Section 2.4.1), which is defined wrt. to both the oracle and algorithm best on average (aka. SBS), we abstain from using it as the SBS cannot be as easily defined as in the standard setting. This is because only the performance of the selected algorithm (and not of all) in the current time step is available to update the underlying model instead of offline training data. The rePAR10 is simply defined as the PAR10 score of the corresponding approach divided by the PAR10 score of the oracle, i.e. the smaller the rePAR10, the better. Moreover, we will explicitly analyze the "prediction time", i.e. the time an approach requires for making a single selection and updating its model with the corresponding feedback.

5.5.1 Ablation Study

First, we analyze how the different LinUCB and Thompson sampling variants perform in terms of rePAR10 performance when some of their components are activated or deactivated.

5.5.1.1 LinUCB

Recall that we differentiate in principle between BlindUCB and BClinUCB. Both the randomization idea (denoted by a 'rand_' prefix) and the expected PAR10 loss minimization (denoted by '_rev' suffix) can, in principle, be incorporated into both, yielding a total of 8 variants.

Figure 5.1 shows the median rePAR10 score over all scenarios of the corresponding variant plotted against its prediction time in seconds. First of all, it is very clear that all of the LinUCB variants are at least 3.1 times as worse as the oracle. A closer look at the selections made by the corresponding models shows two things. First, although BlindUCB heavily underestimates runtimes as it completely ignores censored samples, its estimates yield some of the best algorithm selections among all LinUCB variants. Second, except for the revisited versions, BClinUCB yields worse results than the corresponding BlindUCB variant in all cases. As hinted at earlier, BClinUCB suffers from very large confidence bounds due to the correction, yielding suboptimal selections in many cases. Moreover, one can see that directly

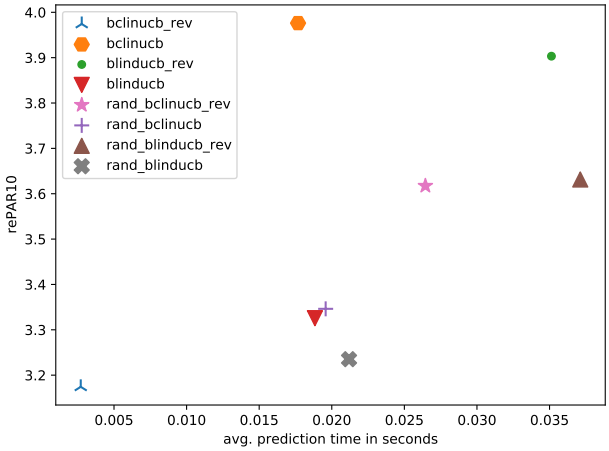

Fig. 5.1: rePAR10 score of the LinUCB variants averaged over all scenarios plotted against their average prediction time in seconds.

minimizing the expected PAR10 loss does not prove to be very beneficial (except for the pure BClinUCB variant) and can even worsen the performance for some variants. From a methodological point of view, this is not surprising for BlindUCB, as it would technically require a different treatment of the expected PAR10 loss based on a truncated (instead of a censored) linear model (cf. Greene [Gre05]). However, the improvement observed for the pure BClinUCB variant, is promising. In fact, when comparing the different selector variants in terms of the number of examples where they yield the best performance, the revisited BClinUCB variant excels (cf. Table A.1). Unfortunately, its performance on the other scenarios is so detrimental that it offsets the other scenarios in the median. In contrast, the randomization (i.e. RandUCB) yields consistent improvements (except for one case), making some of the randomized variants the best among all LinUCB variants. This also coincides with our observation that the poor selection performance is caused by large confidence width due to the correction, which is decreased through randomization.

5.5.1.2 Thompson

We presented both a naïve and a revisited form of Thompson incorporating expected PAR10 loss minimization ('_rev' suffix). Moreover, both versions can be equipped with the Buckley-James imputation strategy discussed at the end of Section 5.4.2 ('bj_' prefix), yielding a total of 4 variants.

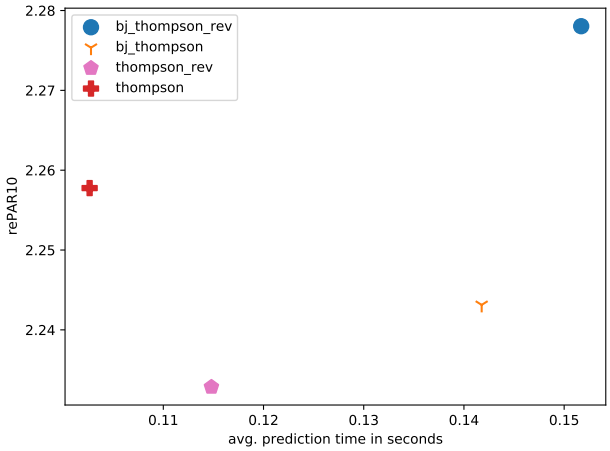

Fig. 5.2: rePAR10 score of the Thompson sampling variants averaged over all scenarios plotted against their average prediction time in seconds.

Figure 5.2 shows the median rePAR10 score over all scenarios of the corresponding variant plotted against its average prediction time per instance. As expected, the more components are active, the longer the prediction time becomes. However, the average prediction time per instance still remains below 0.16 seconds. Both the revisited and the Buckley-James variant yield an improvement over the plain Thompson sampling variant. A combined variant worsens the performance, meaning that the revisited variant achieves the best performance. However, overall, one has to note that all variants behave rather similarly with only small differences in performance.

5.5.2 Comparison to Competitors

In the following, we only compare two UCB and Thompson sampling variants to the competitors to avoid overloading the evaluation. In particular, we compare to an approach by Degroote et al. [Deg+18] (cf. Section 5.6). Their approaches essentially employ batch machine learning models (linear regression or random forests) on the runtime, which are fully retrained after each seen instance. The selection is either done via a simple ϵ-greedy strategy [SB18, Chapter 2] or using a UCB strategy, where the confidence bounds are estimated using the standard deviation extracted from the underlying random forest by means of the Jackknife [SL09] method. In fact, the Degroote approaches cannot be considered true online algorithms due to their dependence on the time horizon — they become intractable with an increasing number of instances. Although one can update the underlying models in principle

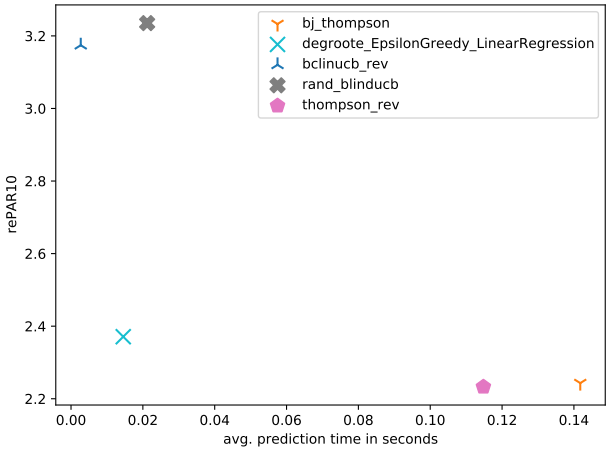

Fig. 5.3: Comparison of Degroote vs. this work in terms of rePAR10 score averaged over all scenarios plotted against their average prediction time in seconds.

less often (e.g., every ten instances as in the original paper), we abstain here from doing so, because our approaches also incorporate every sample immediately.

As we only consider linear models in this work, we only compare to the linear ϵ-greedy strategy presented by Degroote et al. [Deg+18] and abstain from comparing against the random forest versions to avoid that the model complexity becomes a confounding factor in the evaluation.

Figure 5.3 illustrates the rePAR10 value in comparison to the prediction time in seconds of our most successful bandit algorithms and the linear ϵ-greedy Degroote approach. First, it is easy to see that the Thompson sampling variants largely outperform the LinUCB variants in terms of performance at the cost of being slightly slower in terms of prediction time. Second, the Thompson sampling variants improve around 6% in terms of performance upon the Degroote approach. Interestingly, the latter can compete with all online algorithms in terms of prediction time, and even outperforms the Thompson sampling variants. This is mainly because of the limited size of the data, and because the batch linear regression of the library used for the implementation of the Degroote approach is extremely efficient, making batch training affordable. Besides, the Thompson sampling variants require sampling from a multivariate normal distribution, taking up most of the prediction time. Nevertheless, as already said, batch learning will necessarily become impracticable with an increasing number of observations, and sooner or later get slower than the incremental Thompson approach.

Table 5.1 illustrates a more nuanced comparison based on true PAR10 values between the best Thompson sampling variants and Degroote, where the best value for each scenario is printed in bold and the second best is underlined.

Overall, one can verify that Thompson sampling is a much more successful strategy than both ϵ-greedy and LinUCB in OAS in the median. However, the revisited BClinUCB does indeed perform best on many scenarios, although its median performance is decreased by its detrimental performance on the remaining ones. Moreover, directly optimizing the expected PAR10 score (_rev variants) and thereby accounting for the right-censoring of the data often proves beneficial, yielding one of the best OAS approaches in this work in the form of Thompson_rev. Nevertheless, as the large rePAR10 scores indicate, there is still room for improvement.

5.5.3 Sensitivity Analysis

In this section, we provide a sensitivity analysis of the most important hyperparameters of our presented approaches. To keep the number of experiments to perform in a reasonable dimension, we limit ourselves to the most advanced variant of both LinUCB and Thompson sampling we presented in this work. Moreover, we selected six scenarios from the ASlib covering a range of algorithmic problems, number of instances, and features for this analysis. All figures described in the following (Figure 5.4 – Figure 5.8) display the average PAR10 score over ten seeds for different settings of the corresponding hyperparameter. The error bars indicate the corresponding standard deviation.

5.5.3.1 Thompson Variants

Figure 5.4 displays the average PAR10 score over ten seeds for different settings of σ on a selection of scenarios where $\lambda = 0.5$ is fixed. Overall, small values of σ tend to lead to better results indicating that sampling $\widetilde{\boldsymbol{\theta}}_a$, i.e. our belief about the weight vector according to the posterior distribution, should be based on a rather small variance and hence, not too much exploration. This is quite in line with our findings regarding the amount of exploration of the LinUCB variants.

Figure 5.5 displays the average PAR10 score over 10 seeds for different settings of λ on a selection of scenarios where $\sigma = 10$ is fixed. Overall, a clear trend whether

small or large values of λ lead to good results seems hard to detect indicating that the performance is rather robust with respect to the choice of λ.

5.5.3.2 LinUCB Variants

Figure 5.6 displays the average PAR10 score over ten seeds for different settings of σ on a selection of scenarios where $\lambda = 0.5$ and $\tilde{\sigma}^2 = 0.25$ are fixed. In contrast to the Thompson sampling variants previously discussed, where small values of σ tend to lead to better results, here, large values of σ tend to lead to better PAR10 scores indicating that the noise terms defined in Equation 5.4 have a large standard deviation.

Figure 5.7 displays the average PAR10 score over ten seeds for different settings of α on a selection of scenarios where $\sigma = 1$ and $\tilde{\sigma}^2 = 0.25$ are fixed. Overall, no clear trend can be observed whether small or large values of α lead to better results.

Figure 5.8 displays the average PAR10 score over ten seeds for different settings of $\tilde{\sigma}^2$ on a selection of scenarios where $\alpha = 1$ and $\lambda = 0.5$ are fixed. Once again, overall no clear trend can be observed whether small or large values of $\tilde{\sigma}$ lead to good results, due to the wide error bars.

5.6 Related Work

Most related from a problem perspective is the work by Degroote et al. In a series of papers [Deg+16; Deg17; Deg+18], they define the OAS problem in a similar form as we do and present different context-based bandit algorithms. In contrast to their setting, the one presented in Section 2.1.2 does not feature a prior offline training phase, as our goal is to investigate a true online setting where learning has to be performed from scratch. In addition, their approaches essentially rely on batch learning algorithms, making their time- and space-complexity dependent on the time horizon[4]. Moreover, they do not explicitly consider the problem of censoring, but apply a PAR10 imputation (as is standard in ASlib). Lastly, compared to our

[4] As we show in this chapter, some of their batch learning algorithms can actually be replaced by online learners.

work, their approaches lack a theoretical foundation, for instance, their models on the runtimes would in principle even allow negative runtime predictions.

The majority of other work related to OAS is situated in the fields of (online) algorithm scheduling (cf. Section 2.2.1) [LBH16] and dynamic algorithm configuration [Bie+20] (aka. algorithm control [Bie+19]), where the goal is to predict a schedule of algorithms or dynamically control the algorithm during the solution process of an instance instead of predicting a single algorithm as in our case. Gagliolo and Schmidhuber [GS06], Gagliolo and Legrand [GL10], Gagliolo and Schmidhuber [GS10], Pimpalkhare et al. [Pim+21], and Cicirello and Smith [CS05] essentially consider an online algorithm scheduling problem, where both an ordering of algorithms and their corresponding resource allocation (or simply the allocation) has to be computed. Thus, the prediction target is not a single algorithm as in our problem, but rather a very specific composition of algorithms, which can be updated during the solution process. Different bandit algorithms are used to solve this problem variant. Lagoudakis and Littman [LL00], Armstrong et al. [Arm+06], van Rijn et al. [vDB18], Laroche and Féraud [LF17] and Lissovoi et al. [LOW20] in one way or another consider the problem of switching (a component of) an algorithm during the solution process of an instance by means of reinforcement learning or bandit algorithms. They can be considered to be in the field of algorithm control and dynamic algorithm configuration.

Another large corpus of related work can be found in the field of learning from data streams, where the goal is to select an algorithm for the next instance assuming that the data generating process might show a distributional shift [Gam12]. To achieve this, Rossi et al. [RdS12] and van Rijn et al. [van+14] apply windowing techniques and apply offline AS approaches, which are trained on the last window of instances and used to predict for the next instance. Similarly, van Rijn et al. [van+15] dynamically adjust the composition and weights of an ensemble of streaming algorithms. In a way, the methods presented by Degroote et al. [Deg+18] can be seen as windowing techniques where the window size is set to $t - 1$, if t is the current time step.

Another related branch of the literature is real-time algorithm configuration [Fit+14; FMO15; El +20], where in contrast to our setting, one seeks to find a suitable configuration of one single target algorithm (instead of the algorithm itself) for incoming problem instances.

Finally, Gupta and Roughgarden [GR17] analyze several versions of the AS problem on a more theoretical level and show for some problem classes the existence of an OAS approach with low regret under specific assumptions.

5.7 Conclusion and Future Work

In this chapter, we revisited several well-known contextual bandit algorithms and discussed their suitability for dealing with the OAS problem under censored feedback. As a result of the discussion, we adapted them towards runtime-oriented losses, assuming partially censored data while keeping a space- and time-complexity independent of the time horizon. We identified several problems of some of the LinUCB variants in practice, which we solved with further adaptations to some degree. Our extensive experimental study shows that the combination of considering right-censored data in the selection process and an appropriate choice of the exploration strategy leads to better performance. Moreover, as often found in bandit problems, the Thompson strategies yield much better selection performance than the LinUCB variants, presumably because they suffer from less practical problems.

As usual, there exists a plethora of paths of future work to follow. Most importantly, we believe that a generalization of the loss function surrogate models underlying the presented online algorithm selectors to non-linear and potentially tree-based models is a promising path to follow. As mentioned several times, non-linear approaches often achieve much better performance in offline AS, which gives hope that this is also the case in the online setting considered here. Nevertheless, providing theoretical guarantees naturally becomes more difficult for non-linear models. Secondly, a formal regret analysis of the algorithm selectors presented in this chapter is an interesting endeavor. Lastly, investigating whether advanced survival analysis strategies from the streaming community (cf. Section 5.6) can be applied in the context of OAS is also an option worth exploring from our point of view.

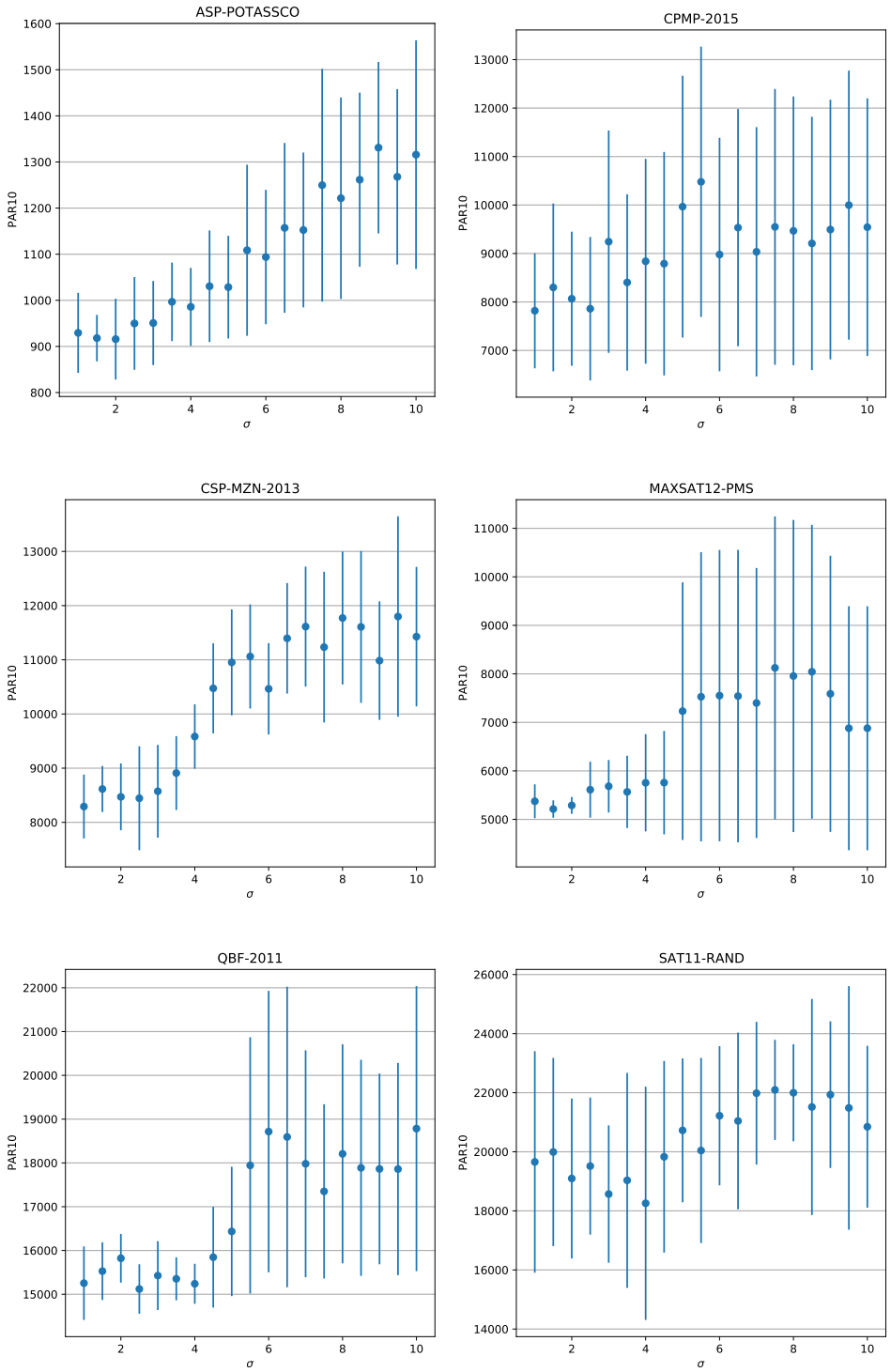

Fig. 5.4: Sensitivity analysis for parameter σ of approach bj_thompson_rev.

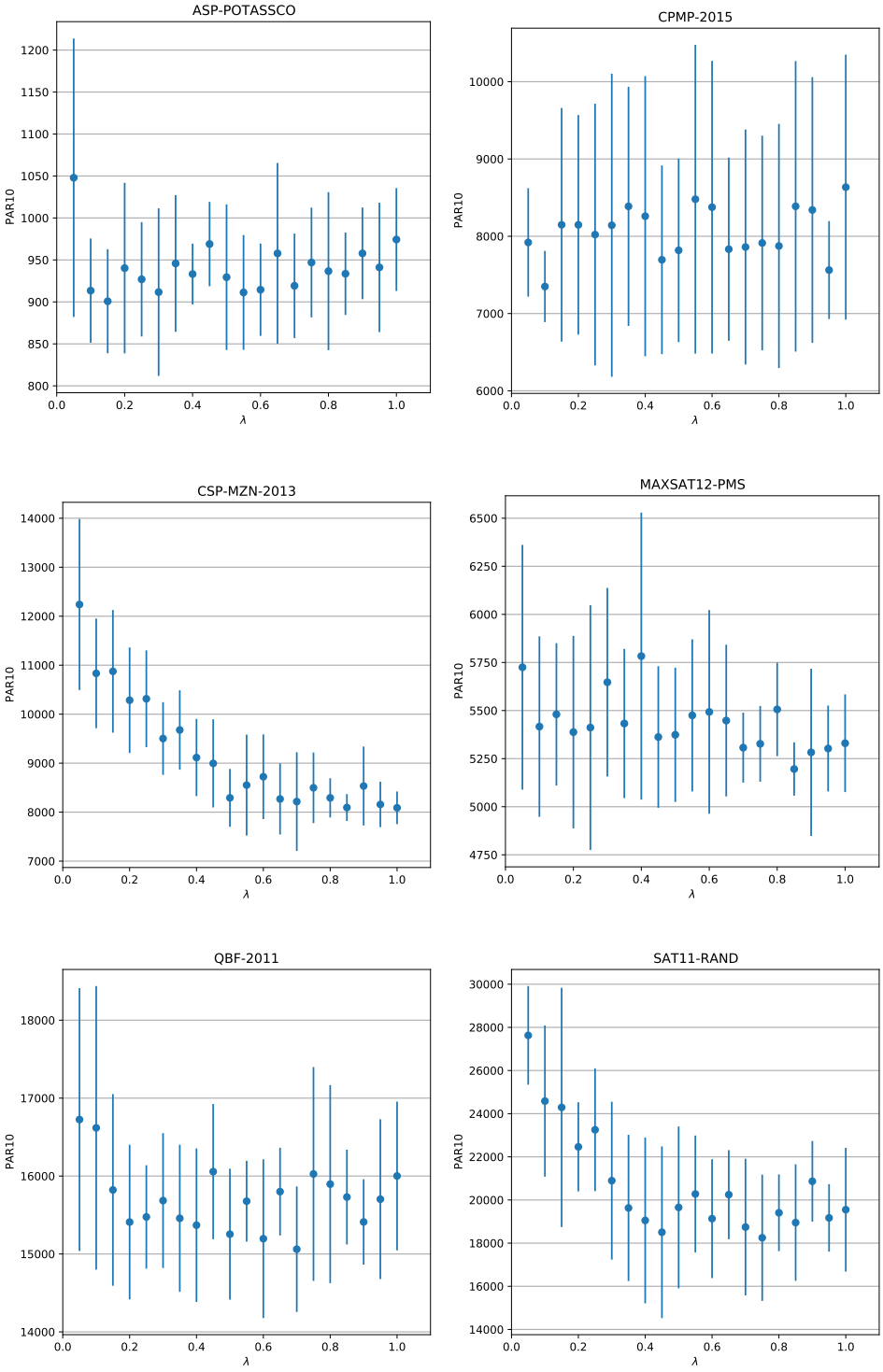

Fig. 5.5: Sensitivity analysis for parameter λ of approach bj_thompson_rev.

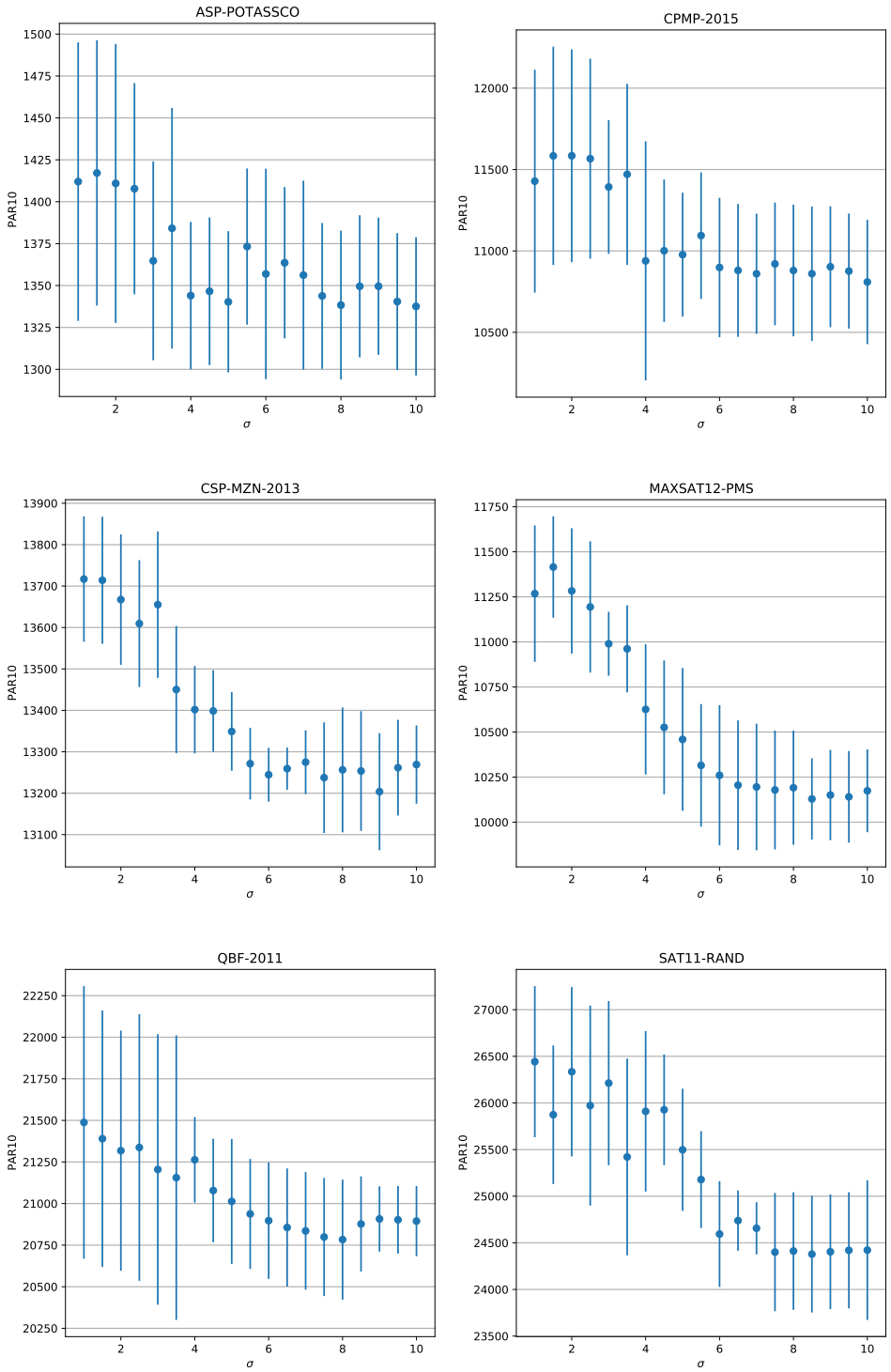

Fig. 5.6: Sensitivity analysis for parameter σ of approach rand_bclinucb_rev.

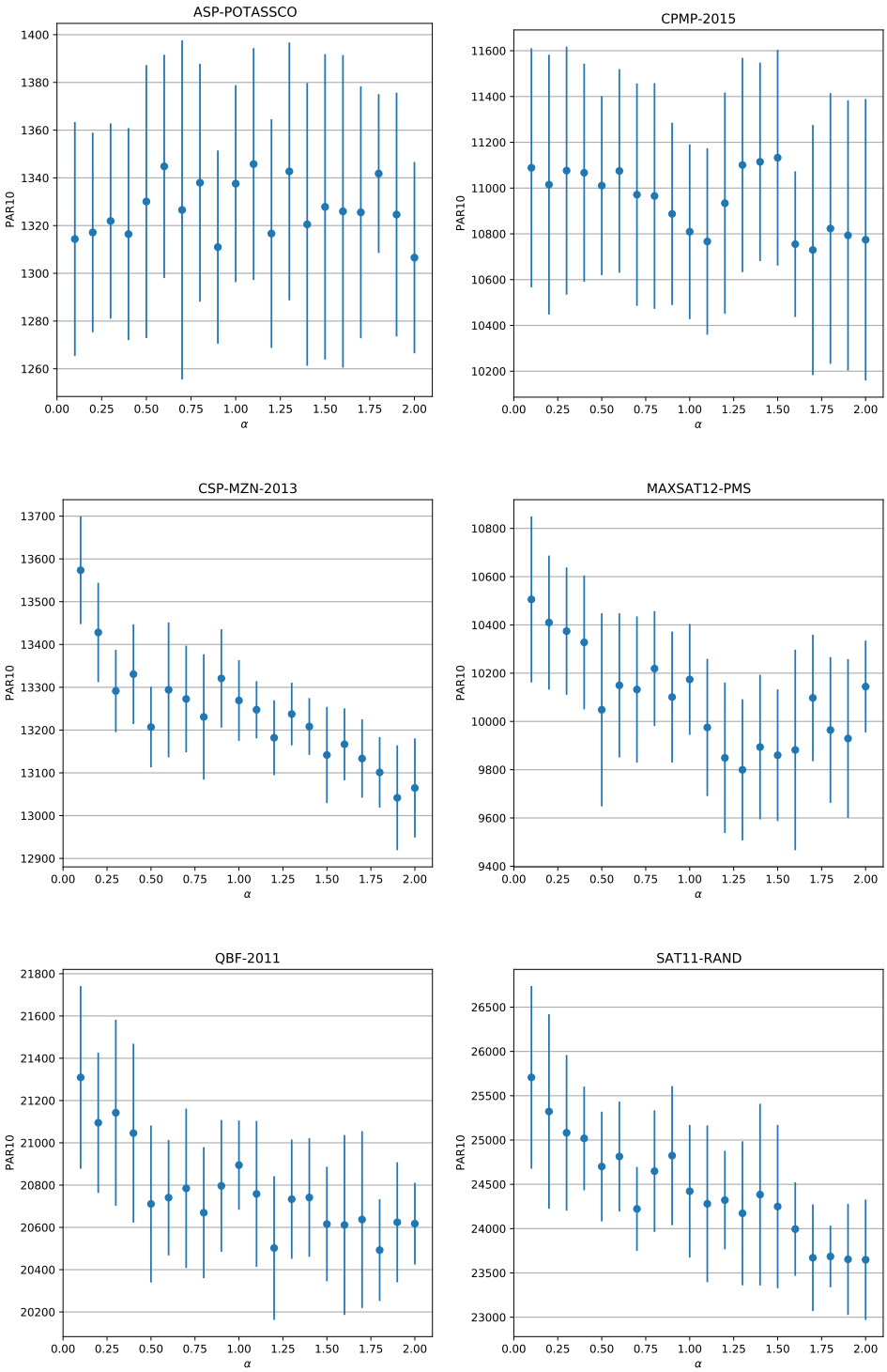

Fig. 5.7: Sensitivity analysis for parameter α of approach rand_bclinucb_rev.

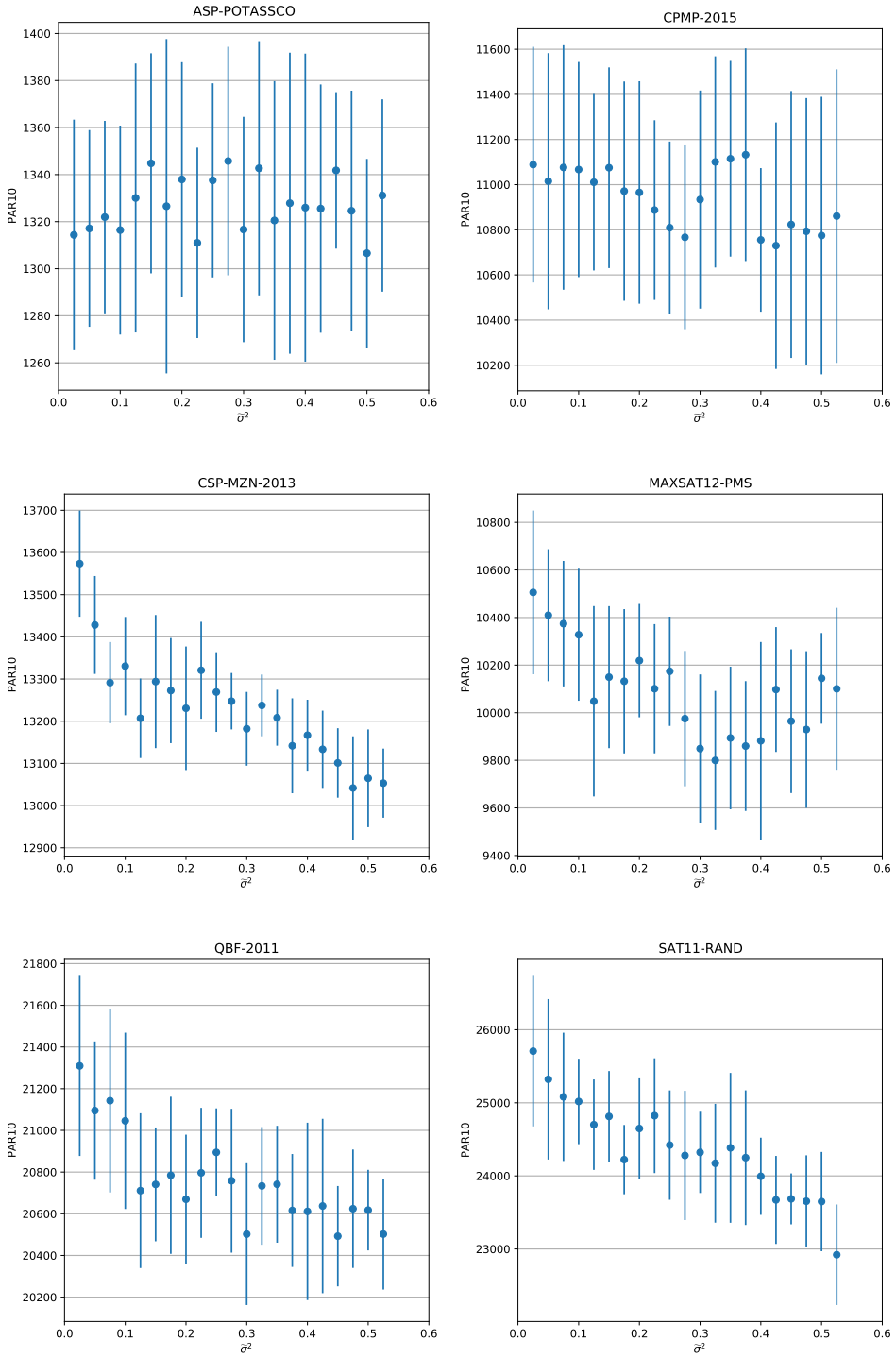

Fig. 5.8: Sensitivity analysis for parameter $\widetilde{\sigma}^2$ of approach rand_bclinucb_rev.

6 Algorithm Selection on a Meta Level

In this chapter of the thesis, we consider the problem of algorithm selection (AS) from a different perspective, i.e. a meta one, based on the observation that not only algorithms themselves can have complementary performance, but selectors as well (Section 6.1). This naturally leads to the meta algorithm selection problem (Section 2.1.3), which is based on the idea of learning when to use which combination of selectors and a corresponding aggregation in order to select a final algorithm. To this end, we present (1) a direct way of meta learning when to select a single selector without the need for any aggregation function (Section 6.2) and (2) a set of approaches based on ensembled algorithm selectors and discuss corresponding limitations. Our extensive experimental study (Section 6.4) corroborates the intuition that leveraging multiple selectors, i.e. ensembling them, proves beneficial compared to meta learning when to select which selector in practice. Furthermore, it highlights that ensembled algorithm selectors are able to beat the already very well performing Run2Survive models presented earlier in this thesis (cf. Chapter 4) by quite a margin. Before concluding this chapter, we embed this chapter into related work (Section 6.5).

In this chapter, we consider no specific loss function, although most of the experiments will be based on scenarios from algorithm selection library (ASlib), which feature runtime as a loss function. Moreover, as already mentioned, we assume the setting of the meta algorithm selection problem defined in Section 2.1.3.

The content presented in this chapter of this thesis has been partly published in the form of a workshop paper [TWH20b] and a journal paper [Tor+22].

6.1 Considering Algorithm Selection on a Meta Level

So far, we have seen that the offline AS problem (cf. Section 2.1.1) and variations thereof can be tackled by a variety of different algorithm selection approaches with rather different underlying principles. While, for example, regression-based AS approaches (cf. Section 2.3.2.2) are trained to predict the concrete loss function value for a specific instance and algorithm pair as accurately as possible and can, in principle, work with any loss function, the Run2Survive approaches presented in Section 4.1 are targeted at optimizing runtime as a loss function and focus on different things depending on the specific variant. Naturally, each of these approaches has strengths and weaknesses, which can prove beneficial on some instances, but also potentially yield rather bad algorithm selections on other instances. Correspondingly, as an end user, choosing the correct algorithm selector can be seen as an AS problem on another level, namely the *meta level*: For a given instance, which algorithm selector from a set of selectors should be used to select the algorithm to run on that instance? The resulting meta-AS problem was first mentioned by Lindauer et al. [LRK19] and Kerschke et al. [Ker+19], though, to the best of our knowledge, without pursuing it further.

One way of answering this question would be through an algorithm selector on the meta level, that is, by an "algorithm selector selector", which does not choose among the algorithms (or "base algorithms", to distinguish them from the AS algorithms), but among the algorithm selectors, which in turn are responsible for selecting an algorithm. However, having access to a set of candidate algorithm selectors, limiting oneself to choosing only *a single* one of them (which in turn chooses the final algorithm) might actually be unnecessarily restrictive. In fact, leveraging a *composition* of selectors, which then choose the final algorithm jointly, might be a better idea. This naturally leads to *ensemble learning* [Die00], which is a common approach in machine learning to combine several predictors into stronger compositions. Thus, instead of using a single algorithm selector to choose an algorithm, a set of selectors is asked to evaluate the available algorithms. Subsequently, these evaluations are aggregated into a joint decision. Somewhat surprisingly, building ensembles of algorithm selectors has hardly been considered in the AS literature so far (cf. Section 6.5), although ensemble learning is well known to improve predictive accuracy in standard machine learning problems such as classification and regression. One reason could be that querying multiple models obviously takes more time than querying only a single one, so that ensembling may appear counterintuitive in scenarios where runtime is considered as the loss function.

6.2 Selecting Single Algorithm Selectors Through Meta Learning

The arguably simplest solution to the meta AS problem is achieved through meta learning [Van18; Bra+08; VGB09], namely to learn which algorithm selector takes the best decision for a given instance. More formally, one could seek to learn a map

$$s_{meta} : \mathcal{I} \longrightarrow \mathcal{S} \,, \tag{6.1}$$

such that the chosen selector returns the most suitable algorithm for a given instance i, i.e.

$$\left(s_{meta}(i)\right)(i) \in \arg\min_{a \in \mathcal{A}} \mathbb{E}\left[l(i,a)\right] \quad . \tag{6.2}$$

In this case, the co-domain of the function ass in Equation 2.7 is effectively restricted to singleton sets $ass(i) = \{s\} \in \mathcal{S}$, consisting of only a single algorithm selector s. We shall discuss the consequences of this self-imposed restriction in Section 6.2.1. Moreover, the aggregation agg in Equation 2.8 is the identity, or, stated differently, there is actually no need for defining or learning an aggregation function. Lastly, the instance features computed by the feature map f for the standard AS problem are also used on the meta level and thus constitute what is known as meta features in the context of meta learning. Likewise, as Equation 6.1 indicates, the set of selectors $\mathcal{S}$ corresponds to the set of meta targets in the meta learning jargon.

Observe that this approach is essentially a special case of the standard AS problem itself, with a very specific set of algorithms to choose from, namely algorithm selectors. Hence, standard AS methods (cf. Section 2.3) can, in principle, be applied. It is important to note that algorithm selection approaches not relying on a feature representation of instances do not necessarily have an advantage in terms of runtime anymore, because they may select an algorithm selector, which in turn requires the feature representation. If the feature computation has to be performed either on the meta or on the base level, its time has to be taken into account as well. However, there is no need to perform the computation twice, if both the algorithm selector and the algorithm selector selector require it, because the resulting features can be shared.

6.2.1 Limits Imposed by Selecting a Single Algorithm Selector

Limiting ourselves to choosing only a single algorithm selector for a given instance, instead of leveraging multiple ones, obviously has consequences in terms of achievable algorithm selection performance. To elaborate on these consequences, let us define an algorithm selector oracle (AS-oracle) as

$$ass^*(i) \in \arg\min_{s \in \mathcal{S}} \mathbb{E}\left[l(i, s(i))\right] \quad . \tag{6.3}$$

It is important to note that the AS-oracle is, in general, not identical to the oracle on the base level, as the set of algorithms to choose from may change. For a better understanding, consider an example with two algorithms a_1 and a_2 and two algorithm selectors s_1 and s_2, where both always select algorithm a_1. Furthermore, assume there exists an instance for which a_2 performs better than a_1, and hence the oracle would select a_2. However, the AS-oracle can only select s_1 or s_2, which in turn both select a_1, resulting in a decrease in oracle performance.

Generally speaking, in order to preserve the original oracle, it is necessary that, for each instance, at least one algorithm selector exists that selects the best algorithm for that instance. Otherwise, the AS-oracle performance may degrade compared to the oracle. In practice, there will be at least one such instance most of the time, and hence an important question is how much the oracle performance degrades. As we show in our experimental evaluation (cf. Section 6.4.2), the degradation strongly depends on the scenario at hand, and ranges from less than 1% to more than 116%[1].

Similarly to the oracle, the single best solver (SBS) on the meta level changes as well, since the single best algorithm selector (SBAS), i.e. the algorithm selector which is best on average, is now an algorithm selector, making it a substantially stronger baseline than the SBS. Hence, while the SBS selects the actual problem-solving algorithm that is best on average and accordingly does not depend on instance features, the SBAS does in fact depend on such features as long as it is not identical to the SBS. Observe that this results in a significant disadvantage for the SBAS in terms of achievable PAR10 scores due to the time required to compute these instance features.

[1] Note that on the CPMP-2015 scenario, the degradation is even around 900%, but constitutes a clear outlier.

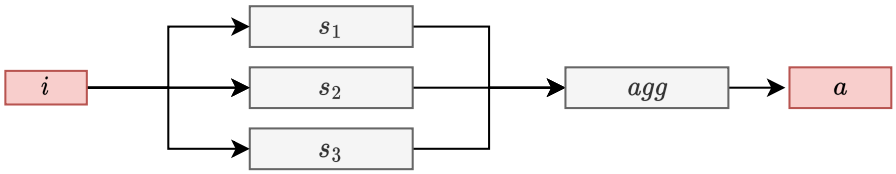

Fig. 6.1: This figure depicts the general process of predicting/selecting an algorithm for a given instance through a trained ensemble of algorithm selectors s_1, s_2, s_3.

Obviously, these implications also influence the performance gains that can be achieved by algorithm selector selectors of the form shown in Equation 6.1 compared to algorithm selectors. As the oracle performance most likely degrades, while the SBS performance most likely improves, the gap between the two also decreases, offering less potential for algorithm (selector) selection approaches to close this gap.

6.3 Constructing Ensembles of Algorithm Selectors

As mentioned earlier, the restriction to choose only a single algorithm selector seems like an unnecessary constraint and may even lead to a potential loss in achievable algorithm selection performance as we have just seen. Accordingly, one may think about using a *composition* of algorithm selectors, which can play to their strengths on some instances while compensating for each other's weaknesses on other instances. This idea motivates us to construct a mapping of the form shown in Equation 2.7 through *ensemble learning*.

Ensemble learning [Die00] presumably constitutes the most natural technique to combine several machine learning approaches into a joint one, with the goal to improve in performance. In algorithm selection, an ensemble can be thought of as a set of algorithm selectors $\mathcal{S}$, called *base algorithm selectors*, which are either trained independently or dependently on each other. At prediction time, each selector is queried for the given instance i, and the algorithm choices are aggregated into a final choice using an aggregation function as defined in Equation 2.8. The concrete strategy used to make the selectors cooperate depends on the ensemble technique being used. Figure 6.1 depicts the general process of predicting/selecting an algorithm for a given instance through a trained ensemble of algorithm selectors.

As mentioned earlier, allowing for the selection of multiple algorithm selectors also requires the definition of an aggregation function in order to finally return a single algorithm. In principle, the aggregation functions can either depend on the instance, i.e. are instance-specific, or can be fixed across instances. Similarly, they can either be learned or predefined.

In general, to be successful, ensembles require a certain degree of heterogeneity of the predictions. Therefore, the different algorithm selectors should not always coincide in their selections. Otherwise, it can easily happen that the majority of predictions made by the base selectors are identical. Hence, in such a situation, the prevalent selector (maybe with slight but negligible variations) dominates the predictions of the entire ensemble, only yielding a computationally more expensive variant of the respective dominating selector. To avoid this problem, most ensemble methods strive for a heterogeneous set of base selectors. This can be achieved through a suitable choice of base selectors given to the method, like, for example, in voting. Alternatively, in the case of methods such as bagging, which only work with a single base selector, different variants of the same selector can be trained on different datasets.

Intuitively, the training and querying of more than one selector might be counter-intuitive in settings where runtime is the loss function to be optimized, as it automatically results in larger runtime. In this regard, it is important to note that the majority of the runtime is required for training the selectors in the ensembles. In offline AS, we assume this training to be performed *offline*, i.e. prior to the actual selection of algorithms. Hence, longer training times do not constitute a real disadvantage, as long as prediction (querying the ensemble members) remains fast, which is the case as most selectors are known to be extremely fast such that even compositions of them are slower, but still fast. We discuss this issue in more detail in Section 6.4.8.2.

In the following, we first elaborate on different aggregation strategies. Although some of these aggregation functions include learnable components, they are fixed across instances, i.e. the aggregation of predictions does not depend on the given instance. Then, we present several ensemble techniques for creating a pool of algorithm selectors, in particular voting [Die00], bagging [Bre96], and boosting [Sch90]. We continue with a discussion of stacking [Wol92], which can be seen as a *learned, instance-specific* aggregation method. As such, it is somehow positioned in-between ensemble and meta learning. Finally, we close this section with a methodological comparison of the presented approaches.

6.3.1 Aggregation Strategies

One of the most natural forms of aggregation in our context is *(weighted) majority aggregation*. As the name suggests, it aggregates the algorithm choices by selecting the algorithm that was selected most frequently, potentially weighting the choices of the selectors differently. This is motivated by the idea that selectors with strong performance should potentially be trusted more than weaker ones. More formally, weighted majority aggregation can be defined as[2]

$$agg_{(w)maj}(i, \mathcal{S}) = \arg\max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} w_s \cdot [\![s(i) = a]\!] , \tag{6.4}$$

where $w_s \in \mathbb{R}^+$ denotes the weight associated with selector s. With $w_s = 1$ for all $s \in \mathcal{S}$, we recover standard majority voting. To obtain proper weights, a plethora of methods are applicable in principle. However, we simply consider the *nPAR10* score of the different base algorithm selectors on the training data in order to determine corresponding weights — conducting a cross-validation on the training data for the same purpose turned out to result in similar performance while being computationally more expensive.

Up to now, we assumed that an algorithm selector only returns a single algorithm. While this is typically true in practice, the majority of approaches internally feature more nuanced predictions, often constituting some kind of loss (or score) for each algorithm in $\mathcal{A}$. Accordingly, instead of using only a concrete algorithm choice as the output of the algorithm selectors, we adapted them to return such nuanced predictions where possible.

More formally, let us assume that each *trained* algorithm selector $s \in \mathcal{S}$ cannot only be evaluated on $i \in \mathcal{I}$, but that it also allows access to $\widehat{l}_s(i, a)$, i.e. to the corresponding internal surrogate loss of each algorithm $a \in \mathcal{A}$. For those approaches where such an estimate cannot be extracted explicitly, e.g., classification-based algorithm selectors, we define dummy losses as

$$\widehat{l}_s(i, a) = \begin{cases} 0 & \text{if } s(i) = a \\ 1 & \text{else} \end{cases} \tag{6.5}$$

[2] $[\![\cdot]\!]$ denotes the indicator function evaluating to 1 if the expression is true, and to 0 otherwise.

for all instances $i \in \mathcal{I}$ and algorithms $a \in \mathcal{A}$. Note that this is also the same approach used in Section 2.3 in order to present the different AS approaches in a unified framework of surrogate loss functions.

With this consideration, aggregations on this more nuanced level of scores instead of the level of final choices can be made. The most straight-forward aggregation function on this level is the *arithmetic mean*, i.e.

$$agg_{avg}(i, \mathcal{S}) = \arg\min_{a \in \mathcal{A}} \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \widehat{l}_s(i, a)\,. \tag{6.6}$$

While conceptually simple, it requires the performance surrogates of the different selectors to approximate the same function. Otherwise, the predictions are incomparable, and averaging is not a meaningful operation. For example, combining the output of a ranking loss function optimized by one selector with the estimated average *PAR10* scores of another does not make any sense. In principle, the arithmetic mean can also be turned into a weighted version as done in Equation 6.4.

In order to be able to aggregate on this more nuanced level while overcoming the weakness of the arithmetic mean, we propose to aggregate *rankings* (rank aggregation) of algorithms constructed from the algorithm scores obtained from the selectors. More precisely, we can assume that each selector s returns a ranking over the algorithms in $\mathcal{A}$ by sorting them in increasing order w.r.t. $\widehat{l}_s(i, \cdot)$, such that the presumably best algorithm is put on the first position in the ranking, the second-best on the second position, etc. Having obtained such a ranking over the algorithms for each selector, they need to be aggregated in order to draw a conclusion and eventually return a single algorithm as the final choice.

A very simple method for rank aggregation is called *Borda count* [Bor84]. Given a ranking of n items, it assigns n points to the top item, $n-1$ points to the second-best, and so forth. This is done for each ranking to be aggregated, and the consensus ranking is obtained by sorting the items (algorithms in our case) in descending order according to their total sum of points. As pointed out by Dwork et al. [Dwo+01], the Borda count has a number of less appealing properties, at least from a theoretical point of view. In particular, it does not satisfy the Condorcet winner criterion [Dwo+01]. Roughly speaking, the Condorcet winner criterion states that an item, which is more often ranked better than any other item when comparing them in a pairwise manner, will not necessarily be ranked at the top position in the aggregated ranking. Nevertheless, its linear time complexity makes it fast to compute. This is in sharp contrast to other rank aggregation techniques that involve intractable optimiza-

tion problems [Dwo+01]. Besides, Borda count comes with provable approximation guarantees for several other aggregation techniques [CFR06]. Overall, it seems to be a good compromise for the case of algorithm selection, where predictions are performed under tight time constraints.

Formally, we can use Borda count as an aggregation function for our setting as follows, where $rank : \mathcal{I} \times \mathcal{S} \times \mathcal{A} \to \mathbb{N}$ returns the rank of an algorithm a in the ranking returned by a selector s on an instance i:

$$agg_{borda}(i, \mathcal{S}) = \arg\min_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} rank(i, s, a) \tag{6.7}$$

Ties are handled by assigning to all tied algorithms the average of the block of ranks they occupy [Saa00]. In practice, ties can only be caused by the dummy scores introduced in Equation 6.5. Therefore, they always occur at the end of the rankings. Theoretically, identical scores of $\widehat{l}(i, \cdot)$ could also result in ties, but this never happened in practice.

While the aggregation techniques outlined above appear to be meaningful in the context of the algorithm selection task, we would like to point out that other aggregation techniques are, of course, conceivable and could be used instead.

6.3.2 Voting

Voting ensembles are presumably the easiest form of ensemble learning: Each algorithm selector in a set $\mathcal{S}' \subseteq \mathcal{S}$ is trained independently of the others on the same training data $\mathcal{I}_D$. At prediction time, all algorithm selectors in $\mathcal{S}'$ are queried, and the predictions are aggregated using one of the previously described aggregation strategies. Figure 6.2 depicts the training process of a voting ensemble.

As we demonstrate empirically, it is important to optimize the ensemble composition, i.e. the set of base algorithm selectors $\mathcal{S}' \subseteq \mathcal{S}$ specifying the ensemble, because the performance of a voting ensemble solely depends on this configurable parameter. Intuitively, a complete evaluation of each possible composition to check the corresponding performance might seem intractable due to the exponential (in $|\mathcal{S}|$) number of compositions. However, in practice, this can be a viable option under certain circumstances. To this end, we hold back a portion of the training data $\mathcal{I}_D$ as validation data $\mathcal{I}'_D \subset \mathcal{I}_D$. Then, all base algorithm selectors can be trained on the

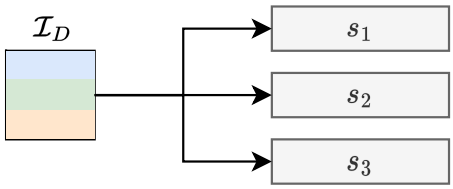

Fig. 6.2: This figure depicts the training process of a voting ensemble, where each base algorithm selector is trained with the same training instances. Ensemble heterogeneity is achieved by choosing a heterogeneous set of algorithm selectors in advance.

reduced training data $\mathcal{I}_D \backslash \mathcal{I}'_D$ once, so that, in order to estimate the performance of an ensemble composition, only the predictions of the used selectors on the validation data $\mathcal{I}'_D$ need to be obtained and aggregated[3]. As the training of the selectors has to be performed only once at the beginning, and the computation of both the predictions and the aggregation can be performed in a negligible amount of time, the evaluation of all possible compositions is feasible as long as the set of algorithm selectors remains moderately large. For example, computing the training performance of each possible voting ensemble composed of up to 7 algorithm selectors required less than 5 minutes for all scenarios presented in Section 6.4. However, we want to stress that this approach still has an exponential complexity even if the corresponding predictions can be obtained quite fast, as the number of ensemble compositions to evaluate is exponential in the number of algorithm selectors. Thus, if the amount of algorithm selectors becomes larger, more sophisticated ensemble pruning methods as by Rokach [Rok09], Lazarevic and Obradovic [LO01], and Hernández-Lobato et al. [HMS09] can be used to find good compositions.

6.3.3 Bagging

In contrast to voting, *bagging* [Bre96], short for "bootstrap aggregating", only leverages a single kind of algorithm (selector). Therefore, heterogeneity between the ensemble members has to be achieved through other means, such as data manipulation techniques. To this end, bagging leverages a data resampling technique from statistics called *bootstrapping*, which works as follows. Given a set of training instances $\mathcal{I}_D$ of size $N = |\mathcal{I}_D|$, it creates a new training instance set by sampling N times from $\mathcal{I}_D$ *with replacement*. The actual ensemble is constructed by sampling k such new training instance sets ${\mathcal{I}_D}^{(1)}, \ldots, {\mathcal{I}_D}^{(k)}$ and training one instantiation of

[3] We note that, although theoretically sound, we do not split up validation data for the ensemble optimization as this resulted in worse performance in practice and thus simply evaluate the performance of a composition on the same training data. Note that despite this, the final evaluation of an approach is still performed on separate test data.

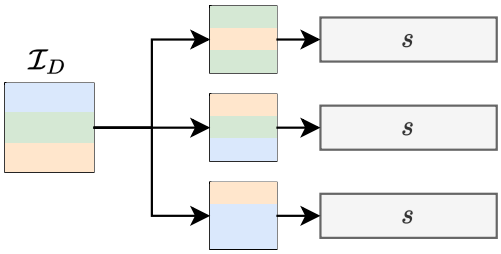

Fig. 6.3: This figure depicts the training process of a bagging ensemble consisting of several instantiations of the same base algorithm selector trained on bootstrapped versions of the original training data.

the provided algorithm selector on each of the k training sets. Thus, the ensemble eventually consists of k algorithm selector instances. At prediction time, one of the previously discussed aggregation functions can be used to aggregate the predictions (selections) of the different selectors. Figure 6.3 depicts the training process of a bagging ensemble.

We would like to point out that we bootstrap on the level of the problem instances and not on the level of the actual training data points ((instance/algorithm)-pairs or (instance/algorithm performance)-pairs). This is done in order to allow the selection algorithms themselves to construct their training data points. In principle, this may lead to differently large training data sets for the corresponding base algorithm selectors if the number of training performance values $l(i, \cdot)$ varies across instances. However, we assume that either $l(i, a)$ is available or we know at least that $l(i, a) > C$ for all $i \in \mathcal{I}_D, a \in \mathcal{A}$, and hence can reasonably impute these missing values, thereby solving the problem of differently sized training data sets.

6.3.4 Boosting

While both voting and bagging fit ensemble members independently of each other (except for (partially) identical training data), boosting *successively* trains its members, each time re-weighting the training instances [Sch90]. After each iteration, i.e. trained selector, the error of the previous selectors is determined and more weight is put onto those instances where a wrong algorithm selection has been performed, while the weight on correctly judged instances is reduced. Similar to bagging, boosting (cf. Figure 6.4) only uses a single selector as a basis of which it trains instantiations based on differently weighted versions of the same training instance set in order to achieve diversity w.r.t. its ensemble members. At prediction

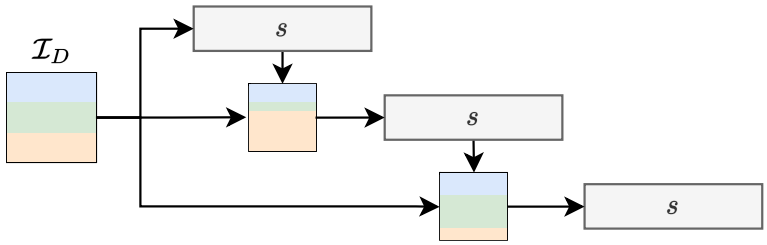

Fig. 6.4: This figure depicts the training process of a boosting ensemble. Similar to bagging, the ensemble comprises several instances of the same base algorithm selector. These are subsequently trained on differently weighted versions of the training data.

time, the predictions of each of the trained selectors are obtained and combined into a joint prediction using a weighted aggregation, using the weights that have been determined as part of the boosting algorithm during the training phase.

In boosting algorithms for multi-class classification, such as SAMME [Has+09], and regression problems, such as AdaBoost.R2 [Dru97], one would naturally consider multi-class classification errors and regression losses, respectively, for re-weighting training instances. However, due to the inferior performance of AdaBoost.R2 in preliminary experiments, we focus on SAMME for the remainder of this paper.

6.3.5 Stacking

In the previous ensemble techniques, the aggregation strategy is always fixed from the beginning and independent of the actual instance at hand. The idea of stacking is to learn the aggregation, i.e. how to best aggregate the predictions of the base algorithm selectors for a given instance. Therefore, a meta-learner

$$\mathbf{h}_{agg} : \mathcal{I} \times \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|} \longrightarrow \mathcal{A} \tag{6.8}$$

is fitted and used to aggregate the predicted losses $\widehat{l}_s(i, a)$ of each algorithm selector $s \in \mathcal{S}$ for a given instance $i \in \mathcal{I}$ and each algorithm $a \in \mathcal{A}$ into a joint decision. To avoid any bias in the training data for the meta-learner, it needs to be ensured that this data is disjoint from the training data of the base algorithm selectors. Therefore, the set of training instances $\mathcal{I}_D$ is normally split into a set of base algorithm selector training instances $\mathcal{I}'_D \subset \mathcal{I}_D$ and a set of meta-learner training instances $\mathcal{I}''_D \subset \mathcal{I}_D$

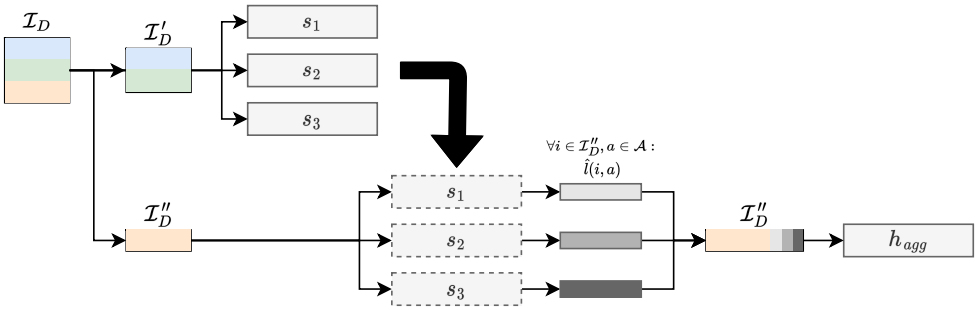

Fig. 6.5: This figure depicts the general idea behind a stacking ensemble. Each ensemble member is trained with the same subset of training instances and the remaining instances are augmented with the corresponding predictions of the trained selectors. Then, a meta-learner, i.e. an additional algorithm selector, h_{agg} is trained on this augmented data, which decides on the algorithm to select.

such that $\mathcal{I}'_D \cap \mathcal{I}''_D = \varnothing$.[4] As all possible base algorithm selectors are used, each can be trained independently on the same subset of training instances $\mathcal{I}'_D$ as a first step such that the training data for the meta-learner can be built. Then, the meta-learner is trained based on the features $\boldsymbol{f}_i \in \mathbb{R}^d$ of each training instance $i \in \mathcal{I}''_D$ extended by the predictions $\widehat{l}_s(i,\cdot)$ of all base algorithm selectors $s \in \mathcal{S}$ on these instances. At prediction time, each base algorithm selector $s \in \mathcal{S}$ is queried, its predictions $\widehat{l}_s(i,\cdot)$ are concatenated and attached to the instance features $\boldsymbol{f}_i \in \mathbb{R}^d$ of instance i, based on which the meta-learner predicts which algorithm to choose. As the meta-learner is an algorithm selector itself, any of the base algorithm selectors can be used. Figure 6.5 depicts the general idea of a stacking ensemble. Note that we only consider a single stacking layer here.

Since stacking is working on an (extended) feature representation, standard feature selection techniques can be used to reduce the number of features and help the meta-learner achieve better prediction performance. Thus, the ensemble composition does not require any optimization upfront. For an overview of feature selection methods, we refer to Guyon and Elisseeff [GE03].

6.3.6 Comparison of the Approaches

To put the approaches presented so far into the broader context of meta algorithm selection (MetaAS), we close this section by revisiting them w.r.t. their most important properties. Figure 6.6 provides an overview and illustrates how the approaches

[4] Although theoretically correct, we did actually not do that split in our experimental evaluation in Section 6.4, as this led to a worse empirical performance.

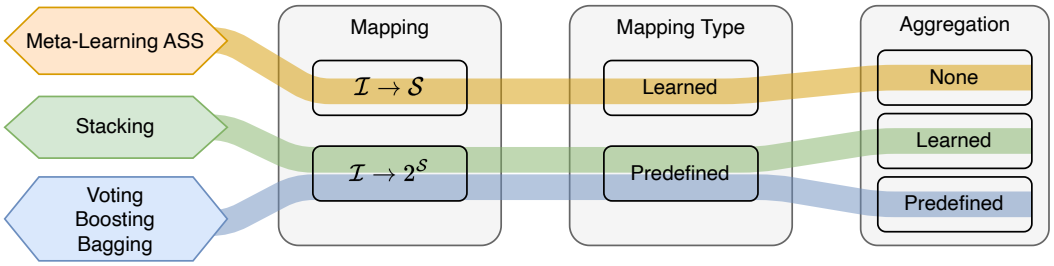

Fig. 6.6: Illustration of the different approaches w.r.t. the kind of mapping they model, how this mapping is constructed, and how the required aggregation is obtained.

relate to each other. It clarifies what kind of mapping these approaches model, how this mapping is constructed, and how the required aggregation function is constructed.

As an important observation, note that some approaches involve learning on the meta level while others do not. The former most obviously holds for learning an algorithm selector selector (cf. Section 6.2), where the modeled mapping is learned directly. On the other side, most ensemble approaches (cf. Section 6.3) do not require any learning on the meta level, because their mapping is essentially predefined. Stacking is somehow in between these two groups: the mapping itself is predefined, but the aggregation function is learned on the meta level.

6.4 Experimental Evaluation

In this section, we provide an empirical evaluation of the ideas presented in the preceding sections. It is organized into four main parts. First, we introduce our experiment setup. Second, we investigate the chance for performance improvements when learning algorithm selector selectors and evaluate the performance of standard algorithm selectors working as algorithm selector selectors. Third, we evaluate the performance of the different ensemble methods presented earlier and discuss the results. We end this section by drawing a broader conclusion from these results.

6.4.1 Experiment Setup

All evaluations are run on a subset of the scenarios from the ASlib v4.0 benchmark suite [Bis+16] (cf. Section 2.6) with a 10-fold cross-validation, where the folds are provided by the benchmark. The list of used scenarios can be inferred from Table 6.1 and Table 6.2.

The performance of the approaches is measured in terms of the *normalized penalized average runtime* (*nPAR10*) metric (cf. Section 2.4.1) if not mentioned otherwise. Recall that a value of 0 indicates oracle performance, values below 1 an improvement over the SBS, and values above 1 a degradation compared to the SBS. To allow for a better visual interpretation, we sometimes illustrate results aggregated over all scenarios. Needless to say, such aggregations have to be treated with care, because (differences between) performance degrees are not easily comparable across scenarios.

The set of algorithm selectors used for the evaluation consists of $\mathcal{S} = \{$PerAlgo, SATzilla'11, R2S-Exp, R2S-PAR10, SUNNY, ISAC, Multiclass$\}$, which have been described in Section 2.3 or used in previous evaluations shown in this thesis. In particular, R2S-EXP and R2S-PAR10 refer to the two Run2Survive variants optimizing the expected runtime and the expected PAR10 detailed in Section 4.4, while the remaining ones have been detailed in Section 4.4.2. These are used both as meta learners, but also as base algorithm selectors for the ensembles. Furthermore, we compare all ensemble variants against the single best algorithm selector, SBAS, in terms of median or mean PAR10 or nPAR10 performance. Lastly, we note that, in general, we leave out instances from the test sets where all algorithms run into the cutoff as no sensible selection is possible for those. However, we do include these instances for the meta learning experiments in Section 6.4.2 as the set of instances in the test sets would otherwise vary between the base level and the meta level yielding incomparable results. This is the case, as we would potentially need to leave out an instance on the meta level (if none of the algorithm selectors chose an algorithm solving it before the cutoff), which we might have included on the base level (since there exists an algorithm solving it before the cutoff time). This problem is very much related to the degradation in oracle performance, which was previously discussed.

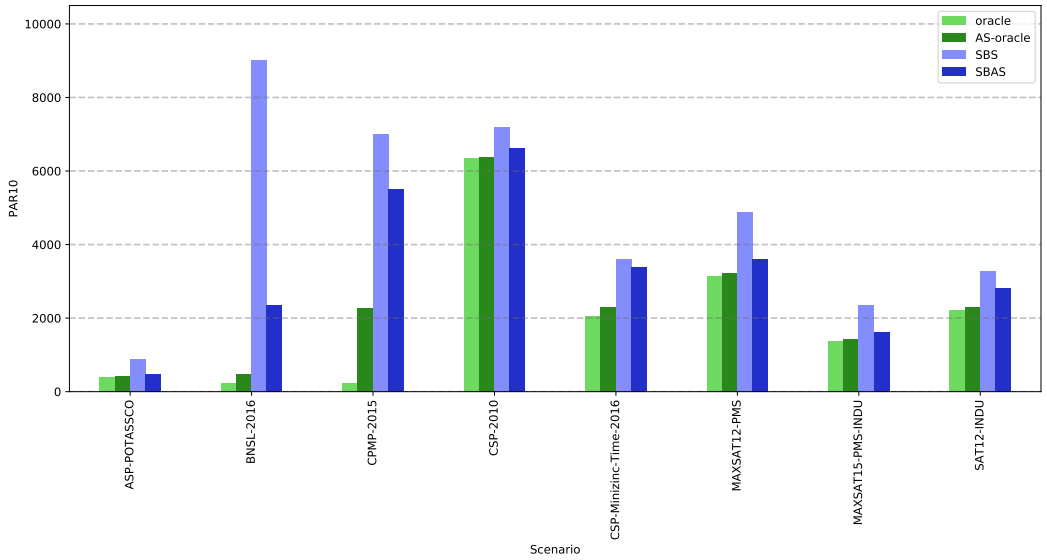

Fig. 6.7: This figure shows the PAR10 scores of the oracle, AS-oracle, SBS and SBAS on a subset of the ASlib v4.0 benchmark scenarios as bar charts.

In the interest of reproducibility of our results, all code, including detailed documentation of the experiments and execution instructions, is available at GitHub[5] and more experiment details can be found in Section A.5.

6.4.2 Meta Learning for Selecting an Algorithm Selector

Figure 6.7 shows the PAR10 scores of the oracle, AS-oracle, SBS and SBAS on a subset of the ASlib v4.0 benchmark scenarios. As one can see, several of the implications we noted in Section 6.2.1 can be validated empirically. Firstly and most importantly, although the SBS/oracle gap is a lot larger than the SBAS/AS-oracle gap, the SBAS/AS-oracle gaps are non-negligible, and hence constructing an algorithm selector selector can in principle make sense. For example, consider scenarios BNSL-2016 or CPMP-2015 with large SBAS/AS-oracle gaps.

As we noted earlier, the reason why these gaps become smaller is that the oracle performance degrades when moving to the meta level for all scenarios, whereas the SBS performance tends to improve, because the SBAS is essentially an algorithm selector. While the degradation in oracle performance is moderate for the majority of scenarios (less than 10%), the improvement of the SBAS over the SBS is non-

[5] `https://github.com/alexandertornede/as_on_a_meta_level`

Tab. 6.1: PAR10 scores of all base- and algorithm selector selectors normalized wrt. the standard oracle and SBS. The result of the best approach is marked in bold for each scenario. Moreover, for the meta-algorithm selectors the values in brackets (a/b) indicate that the approach achieves a performance better or equal to a base-approaches and is worse than b base-approaches.

Level	Algorithm selectors							Algorithm selector selectors (Meta)						
Approach / Scenario	R2S-Exp	R2S-PAR10	ISAC	Multiclass	PerAlgo	SATzilla'11	SUNNY	R2S-Exp	R2S-PAR10	ISAC	Multiclass	PerAlgo	SATzilla'11	SUNNY
ASP-POTASSCO	0.30	0.32	0.60	0.64	0.34	0.47	**0.17**	0.24 (6/1)	0.19 (6/1)	0.24 (6/1)	0.36 (3/4)	0.32 (5/2)	0.31 (5/2)	0.26 (6/1)
BNSL-2016	**0.18**	0.21	0.84	0.31	**0.18**	**0.18**	0.25	0.22 (3/4)	0.21 (4/3)	0.19 (4/3)	0.28 (2/5)	0.22 (3/4)	0.28 (2/5)	0.27 (2/5)
CPMP-2015	0.76	**0.69**	0.90	0.85	0.78	0.70	0.94	0.78 (4/3)	0.78 (4/3)	0.89 (2/5)	0.81 (3/4)	0.77 (4/3)	0.81 (3/4)	0.89 (2/5)
CSP-2010	0.13	0.15	0.31	0.80	0.25	0.13	0.34	0.05 (7/0)	**0.04** (7/0)	0.19 (4/3)	0.13 (7/0)	0.46 (1/6)	0.18 (4/3)	0.09 (7/0)
CSP-MZN-2013	0.11	0.11	0.35	0.31	0.13	0.21	0.13	0.11 (7/0)	**0.10** (7/0)	0.13 (5/2)	0.15 (3/4)	0.13 (5/2)	0.19 (3/4)	0.14 (3/4)
CSP-Mini.-Time-2016	0.43	**0.27**	0.83	0.36	0.67	0.34	0.37	0.51 (2/5)	0.51 (2/5)	0.76 (1/6)	0.60 (2/5)	0.67 (2/5)	0.35 (5/2)	0.51 (2/5)
GLUHACK-18	0.43	0.46	0.69	0.41	0.46	0.42	0.51	**0.40** (7/0)	0.45 (4/3)	0.41 (7/0)	0.49 (2/5)	0.57 (1/6)	0.47 (2/5)	0.46 (4/3)
MAXSAT-PMS-2016	0.60	**0.36**	0.82	1.06	0.77	0.62	0.41	0.64 (3/4)	0.65 (3/4)	0.60 (5/2)	0.71 (3/4)	0.82 (2/5)	0.98 (1/6)	0.75 (3/4)
MAXSAT-WPMS-2016	0.44	0.37	0.76	0.85	0.52	0.31	**0.16**	0.37 (5/2)	0.39 (4/3)	0.62 (2/5)	0.60 (2/5)	0.54 (2/5)	0.43 (4/3)	0.44 (4/3)
MAXSAT12-PMS	0.22	0.23	0.47	0.40	0.28	0.24	0.29	0.25 (4/3)	0.25 (4/3)	**0.20** (7/0)	0.21 (7/0)	0.32 (2/5)	0.22 (7/0)	0.21 (7/0)
MAXSAT15-PMS-INDU	0.34	0.44	0.89	1.06	0.55	0.39	**0.24**	0.36 (5/2)	0.57 (2/5)	0.33 (6/1)	0.39 (5/2)	0.40 (4/3)	0.51 (3/4)	0.26 (6/1)
PROTEUS-2014	**0.41**	**0.41**	0.64	0.84	0.45	0.58	0.47	0.47 (4/3)	0.47 (4/3)	0.48 (3/4)	0.48 (3/4)	0.53 (3/4)	0.62 (2/5)	0.53 (3/4)
QBF-2011	0.21	0.20	0.37	0.35	0.18	**0.15**	0.22	0.20 (5/2)	0.21 (4/3)	0.22 (3/4)	0.21 (4/3)	0.29 (2/5)	0.25 (2/5)	0.26 (2/5)
QBF-2014	**0.26**	0.28	0.51	0.59	0.32	0.31	0.31	0.31 (5/2)	0.30 (5/2)	0.32 (3/4)	0.36 (2/5)	0.41 (2/5)	0.39 (2/5)	0.36 (2/5)
QBF-2016	0.52	0.51	0.65	0.69	0.61	0.61	**0.49**	0.55 (4/3)	0.55 (4/3)	0.52 (5/2)	0.53 (4/3)	0.62 (2/5)	0.57 (4/3)	0.58 (4/3)
SAT03-16-INDU	**0.71**	0.76	0.98	0.99	0.77	0.82	0.82	0.92 (2/5)	0.90 (2/5)	0.80 (4/3)	0.79 (4/3)	0.81 (4/3)	0.84 (2/5)	0.86 (2/5)
SAT11-HAND	**0.34**	**0.34**	0.65	0.57	0.46	0.44	0.60	0.42 (5/2)	0.47 (3/4)	0.42 (5/2)	0.44 (5/2)	0.50 (3/4)	0.45 (4/3)	0.56 (3/4)
SAT11-INDU	0.69	0.69	1.08	0.71	0.63	0.79	0.76	0.78 (2/5)	0.89 (1/6)	0.84 (1/6)	**0.61** (7/0)	0.79 (2/5)	0.73 (3/4)	0.85 (1/6)
SAT11-RAND	0.13	**0.06**	0.59	0.17	0.09	0.39	0.12	0.15 (3/4)	0.12 (5/2)	0.17 (3/4)	0.18 (2/5)	0.18 (2/5)	0.30 (2/5)	0.20 (2/5)
SAT12-ALL	**0.36**	**0.36**	0.67	0.38	0.37	0.44	0.38	0.37 (5/2)	0.40 (2/5)	0.39 (2/5)	0.39 (2/5)	0.40 (2/5)	0.40 (2/5)	0.43 (2/5)
SAT12-HAND	0.34	0.34	0.64	0.41	0.37	**0.27**	0.43	0.34 (6/1)	0.34 (6/1)	0.31 (6/1)	0.38 (3/4)	0.39 (3/4)	0.39 (3/4)	0.38 (3/4)
SAT12-INDU	0.70	0.73	1.02	0.94	0.79	**0.59**	0.78	0.62 (6/1)	0.63 (6/1)	0.75 (4/3)	0.73 (5/2)	0.65 (6/1)	0.65 (6/1)	0.66 (6/1)
SAT12-RAND	0.96	**0.86**	0.91	5.20	1.17	0.93	1.14	0.92 (5/2)	1.02 (3/4)	0.94 (4/3)	1.00 (3/4)	1.25 (1/6)	1.23 (1/6)	1.05 (3/4)
SAT15-INDU	0.95	0.83	0.76	0.91	0.74	0.75	1.00	0.68 (7/0)	0.88 (3/4)	1.00 (1/6)	0.96 (1/6)	**0.65** (7/0)	0.85 (3/4)	0.81 (4/3)
SAT18-EXP	0.61	0.68	0.62	0.65	0.64	0.59	0.63	0.66 (1/6)	0.67 (1/6)	0.61 (6/1)	0.58 (7/0)	0.61 (6/1)	**0.54** (7/0)	0.59 (7/0)

negligible, as the more successful the algorithm selectors considered by the algorithm selector selectors are, the larger this performance gain is.

Table 6.1 shows the *nPAR10* scores of all algorithm selectors and the corresponding algorithm selector selectors of the form shown in Equation 6.1. Moreover, for the algorithm selector selectors, the values in brackets (a/b) indicate that the approach achieves a performance better or equal to a, and is worse than b base approaches.

Unsurprisingly, most algorithm selector selectors are able to consistently improve over the SBS. However, moving to the meta level proves to be beneficial for only seven scenarios and these improvements are even distributed across different algorithm selector selectors. To explain this moderate result, we speculate that the considered AS approaches are not able to unleash their full potential on the meta level, although considerable SBAS/AS-oracle gaps exist, as we have seen previously. However, the win/loss scores in brackets indicate that moving to the meta level is beneficial in the sense that a more robust performance across several scenarios can be achieved.

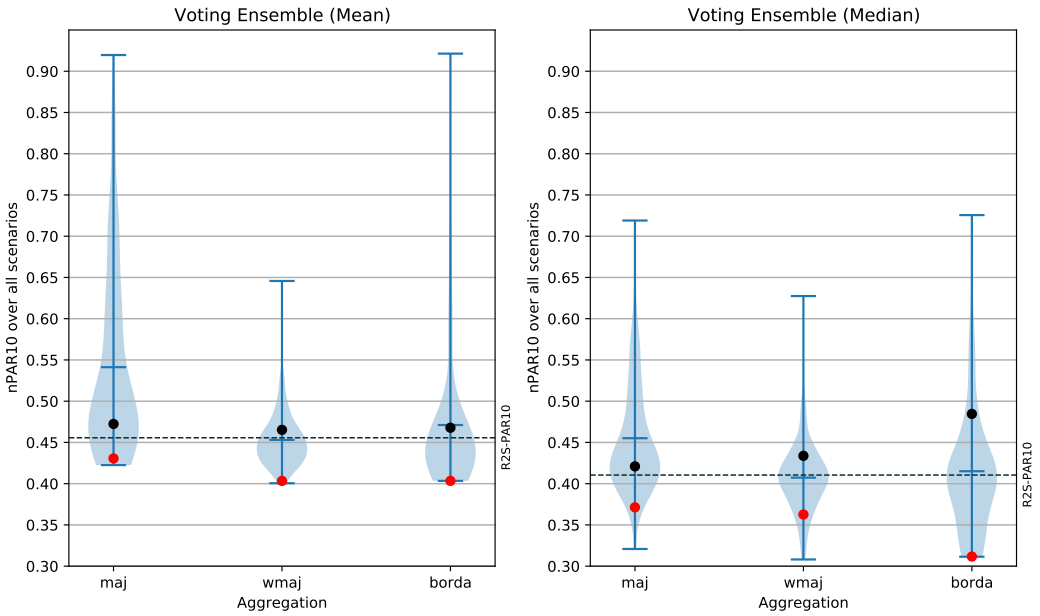

Fig. 6.8: Mean/median performance in terms of *nPAR10* (over all scenarios) of all possible voting ensemble compositions as violin plots grouped by the aggregation strategy being used. The dashed line indicates the performance of the SBAS, the black dot indicates the performance of the best composition w.r.t. the training performance, whereas the red dot indicates the performance of the ensemble with all base algorithm selectors.

6.4.3 Voting Ensembles

Figure 6.8 shows the average/median performance in terms of *nPAR10* (over all scenarios) of all possible voting ensemble compositions as violin plots grouped by the aggregation strategy being used. The dashed line indicates the performance of the SBAS, the black dot indicates the performance of the best composition w.r.t. the training performance, whereas the red dot indicates the performance of the ensemble with all base algorithm selectors.

First of all, it is important to note that voting ensembles offer a lot of optimization potential in terms of both mean and median performance in comparison to the SBAS. While a concrete optimization of the ensemble composition (black dots) does not seem to be beneficial, simply using all possible base algorithm selectors as ensemble members often comes close to the lower performance bound of the voting ensemble strategy. Independent of the aggregation strategy, a voting ensemble with all base algorithm selectors is always able to improve over the best single algorithm selector, sometimes even drastically (e.g., Borda aggregation in terms of median performance). Overall, the weighted majority and the Borda aggregation seem to be

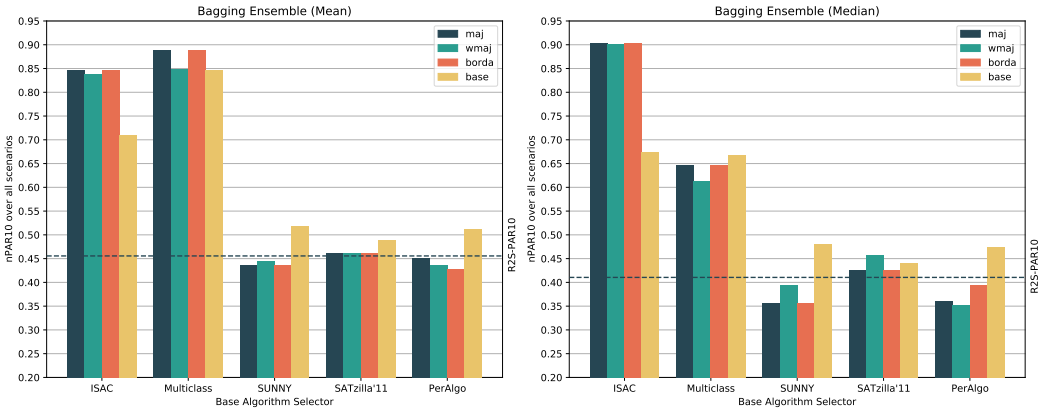

Fig. 6.9: Average / median *nPAR10* performance over all scenarios of each bagging ensemble with 10 instantiations of the corresponding base algorithm selector and different aggregation functions. Moreover, the performance of the corresponding base algorithm selector is shown. Once again, the dashed line indicates the performance of the SBAS.

on par in terms of performance when considering the mean *nPAR10* score, while Borda is superior in the median case.

It is important to understand the scope of the improvement depicted here. Although R2S-PAR10 already offers remarkable performance and represents the state of the art in algorithm selection, it is beaten by around 15% (mean) and 32% (median), which constitute tremendous improvements.

6.4.4 Bagging Ensembles

Figure 6.9 shows the average/median *nPAR10* performance over all scenarios of each bagging ensemble with 10 instantiations of the corresponding base algorithm selector and different aggregation functions. Moreover, the performance of the corresponding base algorithm selector is shown. Once again, the dashed line indicates the performance of the SBAS.

While both ensemble variants equipped with ISAC or Multiclass as a base algorithm selector deteriorate in terms of performance compared to the SBAS, SUNNY, SATzilla'11, and PerAlgo are able to improve both in terms of mean and median performance if the right aggregation is chosen. Surprisingly, none of the aggregation functions seems to be dominating the others. Furthermore, it can be seen that

bagging improves the performance of SUNNY, SATzilla'11 and PerAlgo, but mostly worsens the performance for ISAC and offers mixed results for Multiclass.

In light of the general experience with bagging in machine learning [Bre01; CN06], the performance deterioration of the ISAC ensemble in comparison to its base selector may appear surprising. We conjecture that the negative effect of ensembling is due to the specific characteristics of this method. ISAC applies a clustering technique in order to form clusters over the training instances and computes a threshold t based on the average distances of all instances to their corresponding cluster centroid and the standard deviation over these values. At prediction time, ISAC finds the centroid which is closest to the new instance and returns the algorithm performing best on the cluster, if the distance to the centroid is below the aforementioned threshold. If this is not the case, the SBS is returned. Thus, the threshold can be seen as a fail-safe in case ISAC considers the closest cluster to be too different to draw any reasonable conclusion. After careful investigation, we found that the threshold t decreases for the ensemble members trained on bootstrapped training instance sets as both the average distance and the standard deviation decrease. As a result, the ensemble members mostly deteriorate to the SBS and suggest the SBS on a majority of the instances. This explains the decrease in performance and the similar results of the different aggregation strategies.

We note that Run2Survive was left out as a base algorithm selector for bagging as it cannot easily be trained with bootstrapped instance training sets on scenarios with many censored samples. In such cases, bootstrapping often leads to training data sets consisting of censored samples only, which the approach cannot handle. This is not a problem for the other approaches as they use the standard imputation value for censored samples encoded in the corresponding ASlib scenario.

6.4.5 Boosting Ensembles

Figure 6.10 shows the average/median *nPAR10* performance over all scenarios of each boosting ensemble with 20 iterations and different aggregation functions.

While the performance of the PerAlgo, Multiclass and SATzilla'11 algorithm selectors improves through boosting, the performance of SUNNY and ISAC degrades. Once again, the degradation of ISAC can be explained by the same phenomenon as in the case of bagging: the instance weighting required by boosting was implemented

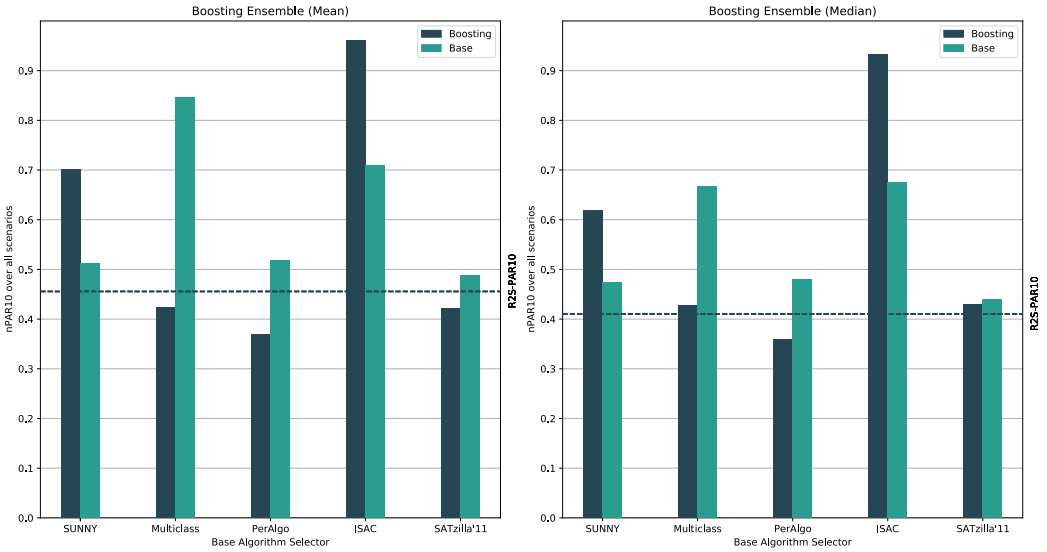

Fig. 6.10: Average / median *nPAR10* performance over all scenarios of each boosting ensemble with 20 iterations and different aggregation functions. Moreover, the performance of the corresponding base algorithm selector is shown. Once again, the dashed line indicates the performance of the SBAS.

through data sampling, whence ISAC mostly degenerates to the SBS. We chose to do so, since not all of the base algorithm selectors inherently support instance weights, but we wanted to investigate boosting variants powered by as many base algorithm selectors as possible. The degradation of the performance of SUNNY can also be explained in a similar fashion. Recall that SUNNY essentially is a similar k-nearest neighbor algorithm, which, given a new instance, returns the algorithm which performs best in terms of PAR10 performance on the k nearest instances in the training data. However, this training data mostly consists of instances with a high weight as all others have a lower chance of being sampled. As a consequence, SUNNY will return the algorithm performing best on average on exactly these instances, while completely ignoring all other instances. This results in degenerate boosting learning curves as depicted in Figure 6.11. The problem is less dominant for selectors that generalize in a more sophisticated way across the features, such as PerAlgo or Multiclass. For instance-based approaches such as SUNNY or ISAC, different forms of boosting specialized for k-NN approaches [GO09] or clustering [FLS04] might be more promising and should be investigated in future work.

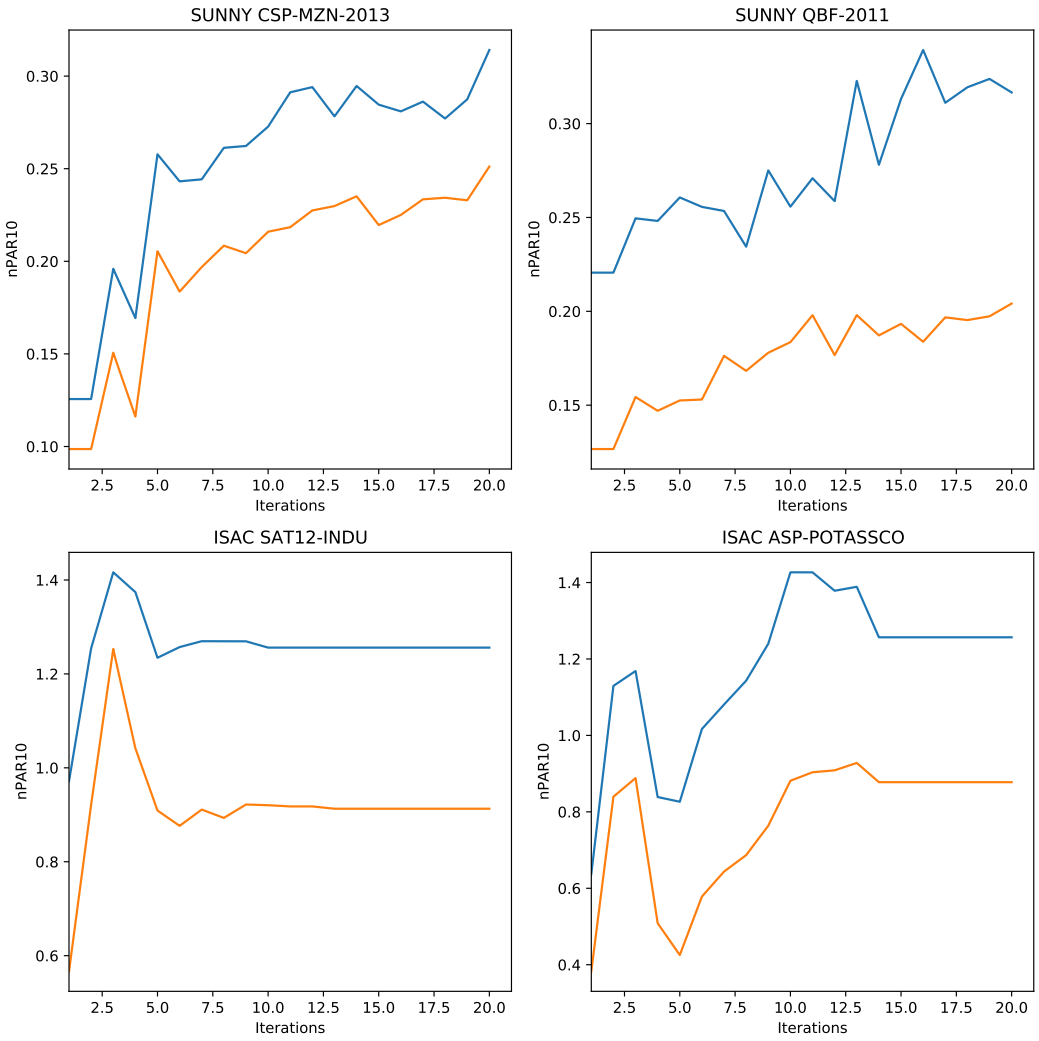

Fig. 6.11: Learning curves featuring training (orange) and testing (blue) *nPAR10* scores of the SAMME boosting algorithm with SUNNY (top two) and ISAC (bottom two) as a base selector on two scenarios.

6.4.6 Stacking

Figure 6.12 shows the average *nPAR10* performance of stacking variants, where the meta-learner h_{agg} is instantiated through different algorithm selectors with and without a variance threshold feature selection approach. Each variant uses all base algorithm selectors to generate additional features. The variance threshold method selects all features with a variance larger than a given threshold, which was set to 0.16 for these experiments. The dotted line indicates the average performance of the SBAS.

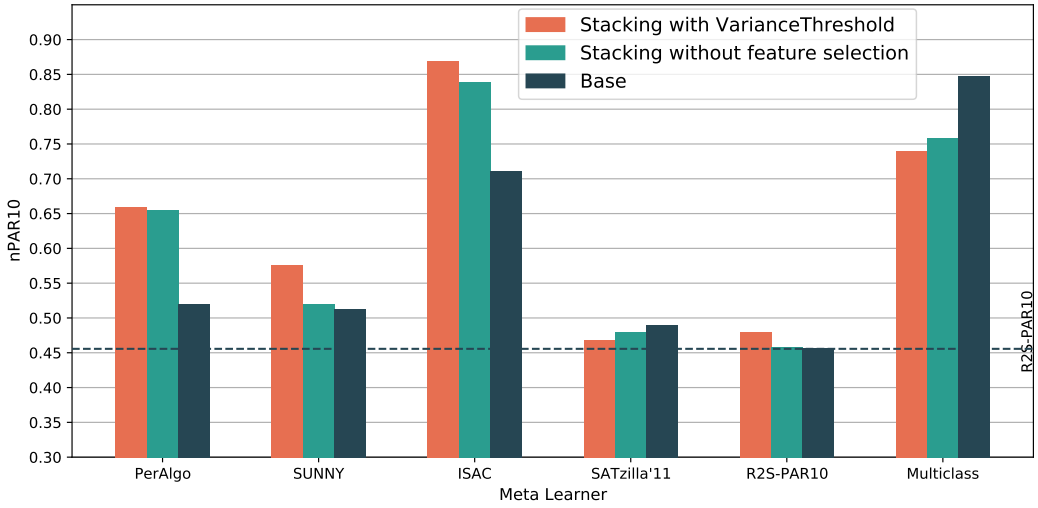

Fig. 6.12: This figure shows the average *nPAR10* performance of stacking variants where h_{agg}, i.e. the meta-learner, is instantiated through different algorithm selectors with and without a variance threshold feature selection approaches.

Firstly, we would like to note that no general recommendation on the use of feature selection can be made, as the effect seems to depend very much on the meta learner. However, while all stacking ensemble variants do not improve over the best single algorithm selector, the variants deploying SATzilla'11 and Multiclass as a meta learner can slightly improve in performance compared to their base versions. We find this quite disappointing because the additional features provided to the meta-learner seem to carry valuable information as the feature importance analysis portrayed in Figure 6.13 corroborates. It shows a ranking over the features w.r.t. their feature importance values extracted from the multi-class classification meta learner (instantiated with a random forest classifier) for the QBF-2011 scenario. Clearly, the additional features in the form of the predictions of the ensemble members carry the biggest part of the information contained in the data.

6.4.7 Overall Comparison

Table 6.2 displays *nPAR10* values of a subset of all evaluated ensemble variants and all base algorithm selectors broken down to the different scenarios. The best result for each scenario is marked in bold, and a line above a result of an ensemble approach indicates that it is better than the result of the best base algorithm selector on the corresponding scenario.

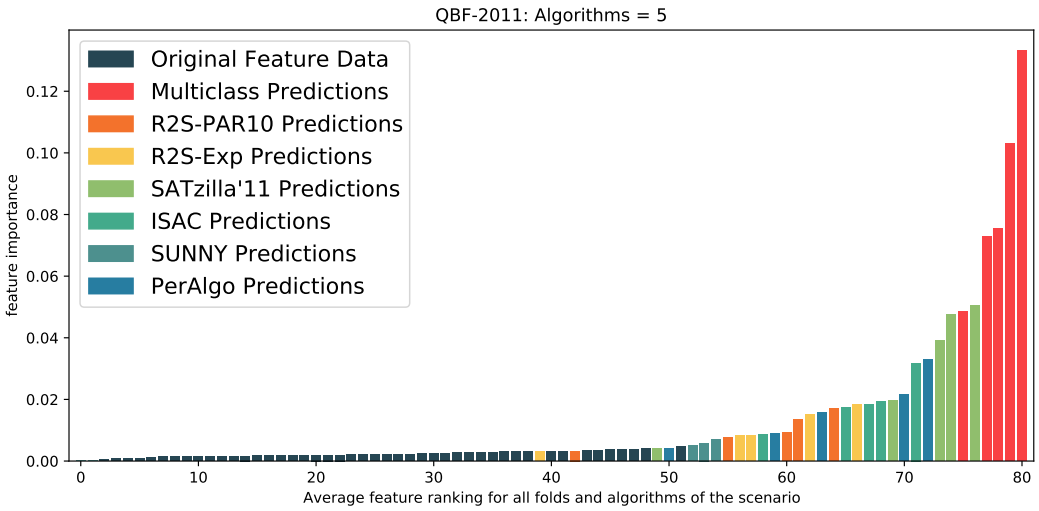

Fig. 6.13: This figure portrays a ranking over the features w.r.t. their feature importance values extracted from the multi-class classification meta-learner (instantiated with a one-vs-all decomposition equipped with a random forest classifier) for the QBF-2011 scenario.

Overall, ensembles of algorithm selectors achieve a performance superior to single algorithm selectors. There are only two scenarios (ASP-POTASSCO, MAXSAT-WPMS-2016) for which none of the selected ensemble variants was able to improve over the base algorithm selector, performing best on that particular scenario, and another three scenarios where a competitive performance was achieved (MAXSAT15-PMS-INDU, SAT11-HAND, SAT12-HAND). For all other scenarios, at least one of the ensemble variants achieved new state-of-the-art performance. While some of these improvements are rather small (CSP-MNZ-2013, where an improvement from 0.11 to 0.10 is recorded), there are also various scenarios with a > 1.5 fold improvement (e.g., CSP-Minizinc-Time-2016, SAT03_16_INDU, QBF-2011). This is especially remarkable as only very few improvements have been made in the last two years.

In terms of median, and average rank performance across all scenarios, the Borda voting ensemble variant achieves the best result and improves over the previous state of the art by more than 32% (median performance). Thus, it demonstrates a very robust performance across all scenarios. The voting ensemble with a Borda aggregation (13), the bagging ensemble with the PerAlgo base selector and a Borda aggregation (11), and the boosting ensemble with the PerAlgo base selector and a weighted majority aggregation (13) all consistently outperform the best single algorithm selector on 11 to 13 of 25 scenarios and, thus, achieve an impressive performance.

Tab. 6.2: *nPAR10* values of the best ensemble variants and all base algorithm selectors broken down to the different scenarios. The best result for each scenario is marked in bold and a line above a result indicates beating all base algorithm selectors.

Ensemble	Voting		Bagging		Stacking		Boosting								
Aggregation	wmaj	borda	wmaj	borda	R2S-Exp	SATzilla'11 (VT)	wmaj	wmaj							
Base selector	all	all	SUNNY	PerAlgo	all	all	Multiclass	PerAlgo	R2S-Exp	R2S-PAR10	ISAC	Multiclass	PerAlgo	SATzilla'11	SUNNY
Scenario															
ASP-POTASSCO	0.26	0.24	0.21	0.29	0.31	0.31	0.44	0.44	0.3	0.34	0.64	0.67	0.34	0.45	**0.17**
BNSL-2016	0.16	0.17	0.25	**0.15**	0.16	0.18	0.32	0.3	0.2	0.22	0.84	0.31	0.2	0.18	0.25
CPMP-2015	0.81	0.87	0.83	0.82	0.88	0.76	0.51	**0.47**	0.97	0.81	0.98	0.94	0.9	0.81	1.05
CSP-2010	0.24	0.24	0.33	**0.23**	**0.23**	0.24	0.43	0.42	0.26	0.26	0.38	0.78	0.36	0.24	0.4
CSP-MZN-2013	**0.1**	0.11	0.12	**0.1**	0.14	0.2	0.39	0.36	0.11	0.11	0.34	0.31	0.13	0.22	0.13
CSP-Minizinc-Time-2016	**0.21**	0.31	0.51	0.51	0.46	0.4	0.39	0.35	0.46	0.46	0.7	0.61	0.61	0.41	0.52
GLUHACK-18	0.44	0.44	0.47	0.49	0.45	0.43	0.4	**0.36**	0.47	0.5	0.6	0.39	0.44	0.41	0.52
MAXSAT-PMS-2016	0.55	0.58	0.39	0.47	0.76	0.7	0.42	**0.38**	0.57	0.41	1.05	1.18	0.79	0.6	0.49
MAXSAT-WPMS-2016	0.34	0.28	0.26	0.33	0.43	0.45	0.41	0.38	0.46	0.38	0.76	0.84	0.49	0.37	**0.24**
MAXSAT12-PMS	0.27	0.27	**0.17**	0.21	0.28	0.33	0.38	0.36	0.27	0.29	0.55	0.37	0.33	0.24	0.28
MAXSAT15-PMS-INDU	0.36	**0.24**	0.31	0.4	0.34	0.3	0.37	0.36	0.39	0.46	1.0	1.24	0.58	0.43	**0.24**
PROTEUS-2014	0.42	0.42	0.43	0.39	0.41	0.58	0.41	**0.36**	0.41	0.41	0.62	0.84	0.45	0.58	0.47
QBF-2011	0.18	0.17	0.16	**0.1**	0.16	0.15	0.39	0.34	0.19	0.19	0.33	0.33	0.2	0.16	0.22
QBF-2014	0.28	0.26	0.36	**0.25**	0.28	0.36	0.4	0.32	0.3	0.31	0.51	0.63	0.31	0.36	0.4
QBF-2016	0.42	0.41	0.44	0.42	0.56	0.63	0.41	**0.33**	0.47	0.49	0.59	0.68	0.65	0.62	0.51
SAT03-16_INDU	0.73	0.71	0.7	0.75	0.66	0.81	0.44	**0.36**	0.72	0.76	0.94	0.99	0.89	0.84	0.85
SAT11-HAND	0.37	**0.36**	0.45	0.45	0.46	0.44	0.43	**0.36**	0.51	**0.36**	0.69	0.57	0.48	0.49	0.7
SAT11-INDU	0.66	0.65	0.97	0.66	0.71	0.69	0.45	**0.38**	0.66	0.74	0.98	0.76	0.62	0.83	0.85
SAT11-RAND	0.1	0.1	0.1	**0.07**	0.11	0.29	0.44	0.37	0.14	0.08	0.61	0.17	0.11	0.36	0.13
SAT12-ALL	0.3	0.3	0.36	**0.29**	0.36	0.38	0.43	0.36	0.37	0.35	0.67	0.37	0.37	0.44	0.4
SAT12-HAND	0.28	**0.27**	0.34	0.3	0.29	0.28	0.43	0.35	0.35	0.34	0.64	0.41	0.38	**0.27**	0.42
SAT12-INDU	0.61	0.58	0.71	0.73	0.73	0.61	0.45	**0.35**	0.73	0.75	0.97	0.94	0.81	0.61	0.81
SAT12-RAND	0.87	0.89	0.91	1.06	0.87	0.9	0.48	**0.37**	1.0	0.9	1.01	5.32	1.18	0.99	1.11
SAT15-INDU	0.65	0.7	0.79	0.85	0.9	0.68	0.5	**0.39**	1.01	0.79	0.72	0.86	0.72	0.72	1.04
SAT18-EXP	0.47	0.52	0.57	**0.38**	0.52	0.59	0.5	0.4	0.6	0.67	0.61	0.65	0.62	0.6	0.63
Mean	0.4	0.4	0.45	0.43	0.46	0.47	0.42	**0.37**	0.48	0.46	0.71	0.85	0.52	0.49	0.51
Median	0.36	**0.31**	0.39	0.39	0.43	0.43	0.43	0.36	0.46	0.41	0.67	0.67	0.48	0.44	0.47
Avg. Rank	4.32	**4.2**	6.92	5.28	6.96	7.56	7.72	5.8	8.2	7.72	13.48	12.88	10.16	8.44	10.32

6.4.8 Discussion of Results

We have seen that ensembles of algorithm selectors achieve a performance superior to single algorithm selectors and are also superior to meta learning an algorithm selector. Considering that the improvements beyond standard algorithm selectors are very considerable, we feel the need to discuss the scope of the results (Section 6.4.8.1), the overhead caused by ensembles of algorithm selectors (Section 6.4.8.2) and whether learning on a meta level might be harder than on the base level (Section 6.4.8.3).

6.4.8.1 Scope of Results

As the composition of the ASlib benchmark and existing literature show, most of the algorithm selection research is centered around constraint satisfaction problems, where the loss to optimize is algorithm runtime or a penalized version thereof such as the PAR10. This has several reasons: First, constraint satisfaction problems play a very important role in industry while, despite the large amount of research committed to this kind of problems over the last century, they remain hard to solve in general. Second, these problems exhibit a high amount of performance complementarity among algorithms, which is the main motivation for the AS problem as discussed earlier. More precisely, algorithms for solving constraint satisfaction problems are known to exhibit heavy-tailed runtime distributions [GSC97], i.e. they need very long to solve some instances while other algorithms might solve the same much faster. Overall, the potential for algorithm selection is very large on this kind of problem, while sometimes lower for other problems such as selecting machine learning algorithms for a dataset. For that particular example, both random forests and gradient boosting often show strong performance and, thus, constitute strong SBS, which in principle can be improved upon as for example shown by Thornton et al. [Tho+13], but often to a smaller degree.

Correspondingly, as is common in the AS literature, the results presented so far focus on scenarios optimizing algorithm runtime. However, in order to at least give an idea about the applicability of the proposed framework for other algorithmic problem classes, we would also like to present results on two other scenarios from ASlib, which focus on optimizing solution quality instead of algorithm runtime. In particular, we present results on the OPENML-WEKA-2017 and the TTP-2016 scenarios. While the former is concerned with the selection of machine learning algorithms for different datasets, the latter deals with selecting algorithms for instances of the traveling thief problem [BMB13]. As the Run2Survive models are specifically tailored towards AS wrt. algorithm runtime instead of performance, we leave them out of the comparison here.

Table 6.3 shows the results for the ensemble methods and the base algorithm selectors including both the SBS and the oracle as reference points. These reference points are included as this table does not show *nPAR10* scores, but a performance score in the unit interval where 1 is the optimum since the scenarios are concerned with solution quality optimization as noted earlier. While the base algorithm selectors are able to achieve a slight improvement over the SBS on the TTP-2016 scenario,

Tab. 6.3: Performance values (OPENML-WEKA-2017: accuracy, TTP-2016: TTP objective function [Wag+18]) of the best ensemble variants and all base algorithm selectors broken down to the respective scenarios. The best result for each scenario is marked in bold and a line above a result indicates beating all base algorithm selectors.

Ensemble	Voting		Bagging		Stacking		Boosting								
Aggregation	wmaj	borda	wmaj	borda	R2S-Exp	SATzilla'11 (VT)	wmaj	wmaj							
Base selector	all	all	SUNNY	PerAlgo	all	all	Multiclass	PerAlgo	PerAlgo	SUNNY	ISAC	SATzilla'11	Multiclass	sbs	oracle
Scenario															
OPENML-WEKA-2017	0.85	0.85	0.85	0.85	0.85	0.85	0.85	**0.86**	0.85	0.84	0.85	**0.86**	0.85	**0.86**	0.88
TTP-2016	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**	0.96	1.0

none of them can beat the SBS on the OPENML-WEKA-2017 scenario. Similarly, none of the ensemble approaches is able to improve over a base selector on the two scenarios. Hence, the empirical results corroborate the essence of the discussion above: On both scenarios, the SBS is a strong baseline, which is quite close to the oracle in terms of performance and hence, there exists hardly any potential for algorithm selection in general, let alone at the meta level[6].

Overall, algorithm selection on the meta level is only sensible in cases where (a) a considerable gap between the performances of standard algorithm selection approaches and the oracle exists and (b) performance complementarity among the algorithm selectors can be exploited. In contrast to the scenarios concerned with runtime, at least the first condition is not met for the two additional scenarios here, making an application not worthwhile for these cases.

6.4.8.2 Runtime Overhead of Algorithm Selector Ensembles

As we are mainly focused on runtime as a loss function in this chapter, one might argue that ensembling multiple algorithm selectors yields an overhead as multiple selectors have to be queried at runtime and thus is not a good idea. Recall, that the PAR10 loss of an approach is mostly dominated by those instances where an algorithm is selected, which runs into a cutoff due to the large penalty of $10 \cdot C$. While

[6] As the ensemble approaches performed much better than the direct meta learning approach in the runtime experiments, we focused on the ensemble variants here.

it is true that the selection time of the ensemble approaches is larger than the time of a single algorithm selector, it is negligible compared to the corresponding PAR10 score. For example, let us consider the first fold of the ASP-POTASCO scenario. Here, the average time an AS approach needs to select an algorithm for a given instance (after the features have already been computed) varies from 0.0009 (SUNNY) to 0.66 (SATzilla) seconds in the considered set of algorithm selectors. Correspondingly, the ensemble methods will require a multiple thereof in addition to some overhead for aggregation and possibly other operations on the meta level. For example, the boosting approach from the evaluation with 10 SUNNY base selectors requires 0.010 seconds whereas voting requires 1.04 seconds. While this does certainly constitute a significant increase in selection time, it is negligible compared to the PAR10 score (which is also the penalized runtime of the selected algorithms averaged across instances) of the fastest algorithm selector on this fold, i.e. R2S-PAR10 with a score of 132 (if instances, for which all algorithms time out, are not considered). Even in cases where we ensemble 10 variants of SATzilla (the selector with the longest selection time), the ensemble requires around 6 seconds, which is less than 5% of the PAR10 score of R2S-PAR10. Thus, the overhead incurred by the ensembles is negligible compared to the PAR10 loss. From a methodological point of view, this observation makes sense as it is well-known in the algorithm selection community that the selection time of most approaches (after the instance features have been computed) is extremely fast and thus negligible. As the time of the ensembling itself is also negligible, the overall time of ensembled selectors remains rather negligible as long as the ensemble does not become too large.

6.4.8.3 Is Meta Learning Harder Than Learning?

Recall our taxonomy of the approaches presented in Figure 6.6, regarding which kind of mapping they model, how this mapping is constructed, and how the required aggregation function is obtained. Drawing an overall conclusion from the results presented in this work, we cautiously conclude that the presumably simpler problem of learning a mapping as in Equation 6.1 from the instances to the set of algorithm selectors yields worse results than solving the presumably more complicated problem of finding both a mapping from instances to a set of selectors and a corresponding aggregation function. While we observed remarkable performance improvements for all ensemble approaches, the meta learning approach could essentially achieve no improvement at all. Although ensembles are known to often yield better results than single approaches and thus an improvement is to be expected, we believe that

the degree of improvement in a well-researched field such as AS is truly remarkable. Moreover, it is surprising that the meta learning essentially fails and hence, classic AS approaches cannot exploit performance complementarity on the meta level.

As a possible reason, note that the meta learning approach heavily relies on the instance features, which are required for learning on the meta level. On the contrary, ensembles of algorithm selectors do not use these features on the meta level directly (except for stacking), but only aggregate the predictions of multiple selectors. Thus, we speculate that the information contained in the features does not allow for an improvement in performance through moving to the meta level, while the predictions of the selectors do carry enough information to do so. This hypothesis is corroborated by the feature analysis conducted as part of the experiments around stacking (cf. Figure 6.13), which indicates that much more information is present in the predictions of the base selectors than in the original instance features. We attribute stacking's ability to perform successful learning on the meta level (aggregation) to the same reason. While stacking was able to achieve improvements, the arguably most simple ensemble approach in the form of voting, which involves no learning on the meta level at all, achieved by far the best results. Overall, learning on the meta level appears to be a very hard problem.

6.4.8.4 Is There a Meta Limit?

As the empirical evaluation indicates, going to the meta level can prove beneficial in terms of algorithm selection performance, but only if done correctly. Naturally, one may wonder if choosing the right approach at the meta level, e.g. the right ensemble technique, can also be automated yielding an even higher level meta meta problem to be solved by an algorithm, which again could be selected, etc. This leads to the problem of infinite regress, which was recently noted in the context of automated machine learning (AutoML) by Hüllermeier et al. [Hül+21]. Considering that solving the problem at the meta level requires resources, e.g. runtime, and thus creates overhead, there naturally is a limit at which increasing in meta level is no longer a reasonable option. However, further investigation of this question is out of the scope of this chapter.

6.5 Related Work

In the following, we give an overview of the most related work regarding the use of ensemble methods in algorithm selection. As mentioned earlier, this work is surprisingly sparse.

In algorithm selection, it is normally assumed that the set of algorithms $\mathcal{A}$ to choose from is predefined, although the composition of this set can have an influence on the selectors. Therefore, Kordík et al. [KCF18] propose to not simply use all available algorithms as a basis to choose from but to employ ensemble techniques in order to construct the set of algorithms to choose from. Thus, they build ensembles on the level of algorithms, whereas we ensemble on the level of selectors with the goal to create a better combined algorithm selector.

Perhaps indeed most related, both Malone et al. [Mal+17] and Kotthoff [Kot12] suggest a stacking approach: First, a regression model is learned per algorithm to estimate the performance on a given instance, and second, the estimated performances are used as input for a multi-class classification model that eventually selects the algorithm. While Kotthoff [Kot12] only uses the outputs of the performance estimators as input of the meta-learner, Malone et al. [Mal+17] use these in addition to the original features. Moreover, Malone et al. [Mal+17] suggest also including uncertainty information obtained from the performance estimators as input for the meta learner. Both variants are very specific instantiations of the general idea presented in this chapter, using stacking as an ensemble technique and a specific selector as a base algorithm selector. While the approach presented by Malone et al. [Mal+17] resulted in the last spot in the open algorithm selection competition of 2017 [LRK19], Kotthoff [Kot12] showed that a performance improvement is possible for a meta learning scenario concerning the selection of machine learning algorithms. In particular, he elaborated that stacking a classifier on top of the pure performance estimation does yield indeed an improvement in most cases over choosing the algorithm based on the performance estimates only.

Lastly, note that ensembling is quite frequently used in modern AutoML tools such as AutoGluon [Eri+20] or auto-sklearn [Feu+15]. However, this is on a conceptually different level. While we ensemble selectors in this work, most AutoML tools ensemble algorithms, i.e. machine learning pipelines. Nevertheless, there exist ideas to also optimize these tools on the meta level similar to what we have in mind with the MetaAS problem. For example, Feurer and Hutter [FH18] present ideas

on how to further automate AutoML by automatically tuning the hyperparameters of AutoML tools. The concept of optimizing an AutoML tool itself also found very successful application in the latest version of auto-sklearn [Feu+22]. Similarly, Lindauer et al. [Lin+15] present the first (and to the best of our knowledge only) fully self-tuning AS system, called AutoFolio, comprising multiple selectors. The idea behind AutoFolio is to automatically select the algorithm selector and configure it based on a given set of instances, i.e. a scenario. While this appears to be very similar to the idea of MetaAS, note that AutoFolio works on the level of scenarios, i.e. optimizes for groups of instances, whereas we select the corresponding selector(s) for each instance, i.e. instance-specifically.

6.6 Conclusion and Future Work

In this chapter, we revisited the problem of AS from a meta perspective. Based on the MetaAS problem and the associated general methodological framework (cf. Section 2.1.3), we considered several concrete learning methods as instantiations of this framework and compared them conceptually and empirically. In an extensive experimental study, we have shown that the MetaAS problem can be solved efficiently with our framework, and that solutions can provide remarkable improvements in performance, often significantly better than the hitherto state of the art. In particular, we find that ensembling algorithm selectors can yield a drastic performance improvement if done correctly while featuring a negligible overhead in terms of selection time. Finally, we embed our results into a broader context, concluding that learning algorithm selector selectors seems to be harder and less promising than defining them through well-established concepts from ensemble learning.

As usual, several lines of future work are conceivable. Firstly, we deem it promising to investigate the idea of MetaAS in the online AS setting as we believe that the online algorithm selection (OAS) approaches presented in Chapter 5 also show a performance complementarity on the instance level. Secondly, it might be worthwhile to investigate more advanced ensembling strategies such as greedy ensemble selection [TPV08], which has proven to be very effective in AutoML tools [Eri+20]. Similarly, as mentioned earlier, more effort could be put into leveraging specialized ensemble techniques for instance-based learning approaches [GO09; FLS04].

7 Conclusion and Future Work

We close this work by concluding remarks and elaborating on opportunities for future work in the field of algorithm selection (AS).

7.1 Conclusion

In this thesis, we elaborated on the problem of algorithm selection: Given an instance from an instance space and a set of algorithms, we want to select the most suitable algorithm for the given instance.

We have investigated several variations of the AS problem, starting with extreme algorithm selection (XAS), where we assume an extremely large set of algorithms and very sparse training data. We argued that existing approaches have trouble coping with the large set of algorithms due to their strategy of learning separate loss function surrogates per algorithm. As a solution, we proposed to leverage a dyadic feature representation of both algorithms and instances, enabling us to learn a single joint model across all algorithms. Our experimental evaluation showed that such a model works particularly well in scenarios of very sparse training data, which is a realistic assumption in the XAS setting.

Moreover, we examined the problem of partially right-censored training data caused by the assumption that algorithms are executed under a timeout. We discussed existing techniques to cope with this problem and pointed out corresponding problems caused by the fact that these methods are built on top of standard AS approaches instead of inherently considering the problem during the design of an AS system. Run2Survive is our attempt at the latter — an AS approach powered by a survival analysis surrogate loss function model allowing one to estimate algorithm runtime distributions learned from partially censored data. The idea to leverage models from survival analysis together with a risk-averse approach to avoid selecting timeouting algorithms yielded an approach substantially outperforming the hitherto state of the art in algorithm selection.

Driven by our success in the offline AS problem, we also worked on the problem of censored training data in an online setting, called the online algorithm selection (OAS) problem — another variation of the AS problem. To meet the demand for online learning in this new setting, we examined the ability of existing stochastic multi-armed bandit approaches to cope with censored data and, similarly to the offline case, found that they exhibit several drawbacks. In order to alleviate these drawbacks, we adapted them in theoretically grounded ways while keeping a time- and space-complexity independent of the time horizon — an unavoidable property for any true online approach. Our evaluation revealed that these adaptations perform slightly better than existing approaches, whose complexity does depend on the time horizon.

Lastly, we took a step back and observed that the performance complementarity among algorithms motivating the AS problem in the first place can also be observed among different algorithm selectors themselves, calling for a meta algorithm selection (MetaAS) problem to be tackled. Taking this observation into account, we attempted to learn for which instance which algorithm selector should be asked to perform a selection, i.e. we tried to learn algorithm selector selectors — an endeavor which turned out to be not very successful. However, based on this failure, we developed the idea to leverage the power of multiple selectors to select an algorithm, i.e. to compose an ensemble of selectors. If composed correctly, such ensembles turned out to be extremely successful (meta) algorithm selectors despite the additional overhead of querying multiple selectors, even substantially beating Run2Survive in terms of selection performance.

7.2 Future Directions for Algorithm Selection

As the reader has seen, we have already discussed some very immediate future work in each of the technical chapters of this thesis. In this section, we want to touch on different topics for future work in a broader context, sometimes connecting different chapters of this thesis. In particular, we discuss ideas for novel settings, conceptual changes to algorithm selection and benchmarking in the following.

7.2.1 Novel Settings

Meta Online Algorithm Selection In Chapter 6 we have seen that ensembling multiple algorithm selectors can prove beneficial in terms of selection performance in the offline AS problem. Naturally, one may wonder if the same also holds for the OAS problem, i.e. whether ensembling online selectors in an online fashion is a reasonable idea. Although this might sound like a simple task at first, there are several important aspects to consider. First, when ensembling multiple selectors in the online setting, not only the selectors themselves have to be learned in an online fashion, but also any component of the ensemble and the aggregation, in case it is not predefined, has to be adapted online as well. For example, for a weighted voting ensemble, both the ensemble composition and the weights for the aggregation need to be adapted online, making the problem significantly harder than in the offline case. Second, one may wonder whether all selectors or only a subset of them should be updated every timestep. On the one hand, updating fewer ones will prove beneficial if the considered loss is based on runtime, since updating the selectors requires time. On the other hand, missing updates of some selectors could have a negative impact on their performance, nullifying the time saved by not updating them. If one decides to update only a subset of the selectors, the natural question of which selectors to update arises. This can be seen as a preselection multi-armed bandits (MAB) problem [BH19; BH20] on the meta level, essentially yielding an algorithm selection problem at the meta meta level: Which preselection MAB algorithm should be chosen? Asking these kinds of questions brings up the problem of infinite regress recently also noted by Hüllermeier et al. [Hül+21], suggesting that going more and more meta can, in principle, be done, but often introduces even higher level problems to solve. Recall that we also shortly discussed this in Section 6.4.8.4.

Extreme Meta Algorithm Selection The approaches for the MetaAS problem presented in Chapter 6 work well, if the set of selectors is of reasonable size. However, when the set of selectors grows very large similar to the set of algorithms in the XAS problem (cf. Chapter 3), additional problems arise. For example, the idea of directly meta learning a selector selector is no longer easily possible for the same reasons as learning algorithm-wise loss function surrogates is problematic in the XAS setting. Correspondingly, one would require features of algorithm selectors in order to learn a selector selector, which generalizes across the selectors and instances. We believe that designing features to characterize an algorithm selector might be even harder than designing features for the algorithms the selector can select due to the higher

level. In particular, selector features most likely depend on the algorithm features. Furthermore, some of the ensembling techniques, such as voting, can no longer work with the complete selector set as the ensemble but need to carefully choose a small subset by special techniques such as greedy ensemble selection [TPV08] in order to avoid increasing the overhead caused by the ensemble.

7.2.2 Conceptual Approach Changes

Hybrid Ranking and Regression Models for AS We have already discussed the idea of combining ranking- and regression-based AS approaches into hybrid ones in Section 2.3.2.4 and also pointed to the work performed by Hanselle et al. [Han+20] motivated by Sculley [Scu10]. Although the results presented in the corresponding work by Hanselle et al. [Han+20] are preliminary and far from the state of the art in AS, we believe that further investigation of this idea is a promising path. As noted in Section 2.3, both ranking and regression AS solutions have advantages and disadvantages, of which most of the latter could be alleviated by a proper hybrid approach. Yet, it is unclear how exactly such a hybrid approach should look like. The idea of Hanselle et al. [Han+20] to learn a model based on a convex combination of regression and ranking loss functions is certainly one possibility. However, as noted in the paper, this possibility also comes with several disadvantages, such as the requirement to scale the two losses into a similar range to avoid that one dominates the other. Considering that the convexly combined hybrid loss function surrogate can also be viewed as an instantiation of a scalarization approach from the field of multi-objective optimization, one could, of course, also pose the problem as a multi-objective problem in the first place. With this in mind, essentially any multi-objective solution technique could be applied to learn a hybrid loss function surrogate. Furthermore, one may wonder whether it is not possible to design a loss function from scratch with both ranking and regression performance in mind. Lastly, motivated by the promising preliminary results of Fehring et al. [FHT22], we deem it worthwhile to further explore the usage of a hybrid loss function within tree-based approaches to model the surrogate loss function.

Transfer Learning AS approaches are usually trained on a specific algorithmic problem, such as SAT, in order to be applied to exactly that problem. That makes the training in general straightforward in the sense that the set of algorithms and the instance features are identical during training and the actual application. However,

this also limits the amount of training data to those data that are available for the corresponding problem. To alleviate this situation, one may wonder whether it might not be possible to leverage transfer learning [WKW16] to train algorithm selectors across different algorithmic problems and varying sets of algorithms. To the best of our knowledge, the only AS work suggesting something in this direction is by Deshpande and Sharma [DS21], who propose to learn an invariant instance feature representation based on the original feature representations across different algorithmic problems. However, they consider the same set of algorithms to select from. Although the work is very short and the evaluation very limited, it demonstrates that transfer learning is, in principle, possible in the AS setting. Moreover, it hints at what is required to perform transfer learning across algorithmic problems and varying algorithm sets: A way to quantify similarity between both instances independent of the algorithmic problem they are originating from and between algorithms. One way of achieving this is to learn an invariant feature representation as suggested by Deshpande and Sharma [DS21]. This also goes hand in hand with our idea to represent algorithms using features presented in Chapter 3. However, one could also think about learning a similarity function directly, which can then help to quantify whether a previously unseen algorithm is similar to an existing one and should thus perform similarly. Nevertheless, this idea still requires a way to represent algorithms as input to the aforementioned similarity function. Moreover, it is also conceivable that the feature free AS approaches summarized in Table 2.4 might be adapted in such a way that they can be trained using instances from various algorithmic problems. This should be rather easy as most of these approaches only require instances to be represented in a specific form as input to the neural network. Thus, as long as one can represent the instances in that form, the actual algorithmic problem they are originating from is not necessarily important. Whether such an approach can actually learn something useful is an important question, though.

Grey-Box Algorithm Selection Most algorithm selection approaches consider algorithms as black boxes and only represent them indirectly via their loss values on some training data. While this might be sufficient to achieve oracle performance under idealized conditions as shown by Malitsky and O'Sullivan [MO14], practical cases suggest that more information about algorithms helps to make better algorithm selections as, for example, seen in Chapter 3. We believe that a fundamental shift from leveraging knowledge about how well (or badly) algorithms perform to how they work internally to achieve that performance is a very promising path to follow. Hence, algorithms should be treated in a grey box instead of black box fashion, as was also recently suggested in the context of algorithm configuration

(AC) [Ana21] and optimization, in general, [Wes16]. Algorithm features, as suggested in Chapter 3, are one way of moving towards such a grey box idea, but many other ideas are conceivable and, in parts already actively researched by the community. Consider, for example, learning curves [Mv22], which give insight into the learning behavior of machine learning algorithms such as inherent inductive biases of algorithms [MRL22; Ruh+23], or exploratory landscape analysis (ELA) [Mer+13], which helps to understand the optimization surface of an algorithmic problem (cf. Section 2.5.2.2). For this reason, the AC community even suggested transferring the general AC problem setting to a grey box one, called dynamic algorithm configuration (DAC) [Bie+20], where the configuration of an algorithm can be changed during its run based on information regarding the current solution and algorithm state.

7.2.3 Benchmarking

Although algorithm selection library (ASlib) is the standard benchmark for AS and is certainly a fantastic asset for the AS community, it covers only 14 different algorithmic problems[1], and many of these problems are similar to some extent (e.g. MAXSAT and SAT). Correspondingly, many existing AS approaches are only tested on a small set of algorithmic problems or potentially even only a single problem if the approach was developed for a particular problem and thus not tested on ASlib. We believe that increasing the diversity of ASlib in terms of different algorithmic problems is an important endeavor that we, as a community, should pursue in order to enable the construction of general AS approaches, which work well under varying algorithmic problems. This is especially important considering the rise of feature-free AS approaches (cf. Table 2.4) as they promise problem-independent algorithm selection solutions under certain assumptions. As of now, there is no large-scale evaluation of those approaches on different algorithmic problems.

7.3 Thesis Contribution and Impact in a Nutshell

In this thesis, firstly, we developed a taxonomy of algorithm selection approaches based on loss function surrogates, which allows one to formally define those ap-

[1] as of April 28, 2023

proaches in a unified framework. Secondly, we investigated the case of an extremely large set of algorithms, including a discussion of weaknesses of existing approaches, and suggest leveraging a dyadic feature representation to overcome these. Thirdly, we designed Run2Survive — an approach allowing to inherently learn from partially censored training data and incorporating a risk-averse selection strategy for runtime-oriented losses. Fourthly, we considered how to handle censored data in an online AS scenario leading to theoretically grounded bandit-based OAS approaches with a time- and space-complexity independent of the time horizon. Lastly, we examined the AS problem on a meta level by trying to select among or compose algorithm selectors yielding state-of-the-art meta algorithm selection approaches based on the idea of ensembling.

We expect that the contributions made within this thesis have the potential to drastically improve the AS workflow from the point of view of a practitioner. First, due to our algorithm improvements with respect to censored data (Chapter 4 and Chapter 5), practitioners have to spend much less time collecting training data for their AS system, which in addition positively impacts the environment, since fewer algorithm runs have to be performed. Second, generalizing AS to extremely large algorithm sets (Chapter 3) removes to burden from the practitioner to carefully pre-select algorithms to choose from, such that the overall process becomes less error-prone. Similarly, leveraging our work on meta algorithm selection (Chapter 6), practitioners no longer need to choose from tens to hundreds of algorithm selectors carefully comparing their advantages and disadvantages but can simply combine their power. Since AS can be employed for virtually any computationally hard problem, the improvement of its workflow can impact a large number of professionals in practice.

Appendix A

A.1 Details on the Experimental Evaluation of Chapter 3

All code, including detailed documentation of the experiments and execution instructions, is available at GitHub[1].

A.1.1 Hardware

All experiments were run on nodes with two Intel Xeon Gold "Skylake" 6148 with 20 cores each and 192 GB RAM.

A.1.2 Software

All experiments are based on Java 11 implementations and the result collection on a previous version of the PyExperimenter [Tor+23]. A complete list of used libraries and the corresponding version number can be found online[2].

Both the DR and DFReg approach were implemented in Java, mainly using AILibs[3], WEKA[4] and Deeplearning4j[5] as external libraries. All other approaches were implemented in Java, mainly using AILibs[6] and WEKA[7] as external libraries. The Alors implementations internally leverage the original CoFi$^{\text{RANK}}$ implementation[8], which is written in C++.

[1] https://github.com/alexandertornede/extreme_algorithm_selection
[2] https://github.com/alexandertornede/extreme_algorithm_selection/blob/master/ecai-2020-experiments/build.gradle
[3] https://github.com/starlibs/AILibs
[4] https://www.cs.waikato.ac.nz/ml/weka/
[5] https://deeplearning4j.konduit.ai/
[6] https://github.com/starlibs/AILibs
[7] https://www.cs.waikato.ac.nz/ml/weka/
[8] https://github.com/markusweimer/cofirank

A.1.3 Hyperparameter Settings

- DR (parameters of PLNet)
  - learning rate: 0.001
  - hidden nodes: $50, 50, 30, 20, 20$
  - activation function: sigmoid
  - epochs: 0 (only stop using early stopping)
  - minibatch size: 20
  - early stopping: true
  - early stopping interval: 1
  - early stopping patience: 20
  - early stopping train ratio: 0.8
- DFReg (parameters of underlying random forests)
  - Default WEKA parameters
- PAReg
  - regressor: random forest with WEKA standard hyperparameters
- Alors (NDCG)
  - Latent feature predictor: random forest with WEKA standard hyperparameters
  - CoFi$^{\text{RANK}}$ parameters: see [9]

[9] https://github.com/alexandertornede/extreme_algorithm_selection/blob/master/ecai-2020-experiments/src/main/java/de/upb/isml/tornede/ecai2020/experiments/alors/matrix_completion/cofirank/CofiConfig.java

- Alors (REGR)
  - latent feature predictor: random forest with WEKA standard hyperparameters
  - CoFi^{RANK} parameters: see [9]
- RandomRank
  - None
- AvgPerformance
  - bayesian averaging: false
- 1-NN LR
  - k: 1
  - bayesian averaging: false
  - distance function: euclidean
- 2-NN LR
  - k: 2
  - bayesian averaging: false
  - distance function: euclidean

A.2 Details on the Experimental Evaluation of Chapter 4

All code, including detailed documentation of the experiments and execution instructions, is available at GitHub[10].

[10] `https://github.com/alexandertornede/algorithm_survival_analysis`

A.2.1 Hardware

All experiments were run on machines featuring Intel Xeon E5-2695v4@2.1GHz CPUs with 16 cores and 64GB RAM.

A.2.2 Software

All experiments are based on Python 3 implementations. A complete list of used packages and the corresponding version number can be found on Github[11].

All variants of Run2Survive were implemented with scikit-survival[12], scipy[13] and ax[14]. All other approaches were implemented with scikit-learn[15] and scipy[13].

A.2.3 Hyperparameter Settings

The following hyperparameters were used. We first note the hyperparameters shared by all Run2Survive variants under Run2SurvivePAR10 and then changes and additional hyperparameters below the corresponding variant. If a hyperparameter is not explicitly noted, it is set to the corresponding default by the package, the code originates from.

- Run2SurvivePAR10 and Run2SurviveExp (mostly parameters of the underlying survival forest):
  - number of estimators: 100
  - min samples split: 10
  - min samples leaf: 0.0

[11] https://github.com/alexandertornede/algorithm_survival_analysis/blob/master/survival_tests/singularity/survival_analysis_environment.yml
[12] https://scikit-survival.readthedocs.io/en/stable/
[13] https://scipy.org/
[14] https://ax.dev/
[15] https://scikit-learn.org

  - max features: sqrt
  - bootstrap: true
  - oob score: false
- Run2SurvivePoly/Log:
  - min weight fraction leaf: 15
  - α: determined automatically via hyperparameter optimization (HPO)
  - β: determined automatically via HPO
- PerAlgorithmRegressor (RandomForestClassifier parameters)
  - estimators: 100
- MultiClassSelector (RandomForestClassifier parameters)
  - estimators: 100
- ISAC*like (G-Means parameters)
  - min samples: 0.001
  - significance: 0.05
  - initial n: 1
  - final n: 5
- SATzilla'11*like (RandomForestClassifier parameters)
  - estimators: 99
  - max features: log
- SUNNY*like (KDTree parameters)

- leaf size: 30
- metric: euclidean

A.3 Details on the Experimental Evaluation of Chapter 5

All code, including detailed documentation of the experiments and execution instructions, is available at GitHub[16].

A.3.1 Hardware

All experiments were run on machines featuring Intel Xeon E5-2695v4@2.1GHz CPUs with 16 cores and 64GB RAM, where each approach was limited to a single core.

A.3.2 Software

All experiments are based on Python 3 implementations. A complete list of used packages and the corresponding version number can be found on Github[17].

All of our presented approaches (LinUCB and Thompson variants) were implemented in Python by using scipy[13] and numpy[18]. We re-implemented the Degroote approach using scikit-learn[15] in Python. In particular, the linear model is implemented using the LinearRegression estimator from scikit-learn.

[16] https://github.com/alexandertornede/online_as
[17] https://github.com/alexandertornede/online_as/blob/main/online_as_code/anaconda/online_as.yml
[18] https://numpy.org/

A.3.3 Hyperparameter Settings

If not stated differently at the beginning of the corresponding experiment (e.g., sensitivity analysis), the following hyperparameters were used:

- Thompson variants
  - σ: 1.0
  - λ: 0.5
- LinUCB variants
  - λ: 1.0
  - α: 1.0
  - σ: 10.0 (for the _rev variants)
  - $\tilde{\sigma}^2$: 0.25 (for the rand_ variants)
- Degroote Epsilon-Greedy linear regression
  - ϵ: 0.05
  - the hyperparameters of underlying models from scikit-learn were set according to their default values

The values of the hyperparameters of our methods were chosen according to a hyperparameter sensitivity analysis (cf. Sec. 5.5.3). The value of the hyperparameter ϵ for the Degroote approach is as suggested by Degroote et al. [Deg+18].

A.3.4 Caveat

All Thompson variants rely on sampling from the multi-variate normal distribution, which we implemented using the 'np.random.multivariate_normal' method. Unfortunately, this method seems to have a bug, which is caused by the underlying

BLAS implementation of the corresponding SVD, which is performed as part of the method. Changing to various versions of numpy and BLAS did not solve the problem for us. As a consequence, some of the repetitions of the experiments of some scenarios did not complete for some Thompson variants. Below one can find a table indicating how many repetitions are *missing* for the corresponding variant on the corresponding scenario. However, due to the very few amount of data points missing, we do not assume a relevant change for the results.

Scenario	Approach	#Missing seeds
CSP-MZN-2013	bj_thompson	2
PROTEUS-2014	bj_thompson	1
PROTEUS-2014	thompson_rev	1
PROTEUS-2014	thompson	2
SAT03-16_INDU	bj_thompson_rev	2
SAT03-16_INDU	bj_thompson	1
SAT03-16_INDU	thompson_rev	1
SAT03-16_INDU	thompson	3
SAT12-RAND	bj_thompson_rev	1
TSP-LION2015	bj_thompson	1

A.3.5 Detailed Performance Data

Table A.1 shows the average PAR10 scores (averaged over 10 seeds) and the corresponding standard deviation of all discussed approach variants and all Degroote variants for reference. Once again, the best value for each scenario is printed in bold, whereas the second best is underlined. As elaborated on earlier, the Thompson variants achieve the best performance.

In order to represent the performance of our approaches in a more detailed way than is the case in Table A.1, we have plotted in Figure A.1 the averaged cumulative PAR10 regret curves (regret wrt. the oracle) of the best Thompson, the best LinUCB and the Degroote approach along with their standard deviation. Here, the cumulative regret up to time T of an approach s_{online} is defined as $\sum_{t=1}^{T} l(i_t, s(h_t, i_t)) - \sum_{t=1}^{T} l(i_t, s^*_{online}(h_t, i_t))$, where s^*_{online} is the oracle and l as in Equation 5.1 with $\mathcal{P}(C) = 10C$. It is not difficult to see that LinUCB cannot compete with the other approaches in many cases and also features a much larger standard

deviation than the others. However, in several cases such as Figures A.1o, A.1r, or A.1z, the differences become much more subtle. Comparing the Thompson variant with the Degroote approach, we see that the former is at least competitive with the latter on almost all scenarios, while being even better on some scenarios (e.g. Figure A.1b, Figure A.1k, Figure A.1m, Figure A.1o). Of course, there are also a few scenarios where the Degroote approach performs better (e.g. Figure A.1n).

A.4 Theoretical Additions to Chapter 5

A.4.1 Deriving the Bias-Corrected Confidence Bounds

In this section, we present details on the derivation of the bias-corrected confidence bounds used in Section 5.3.1. For better readability, let $\boldsymbol{x}_t = \boldsymbol{f}_{i_t}$ be the instance feature vector at timestep t in the following. In particular, we focus on the solution of Equation 5.9, i.e. the RR estimate of the weight vector for the linear loss function surrogate learned with imputed data:

$$\widehat{\boldsymbol{\theta}}_{t,a} = \underset{\boldsymbol{\theta}\in\mathbb{R}^d}{\arg\min} \sum_{j=1}^{t} [\![a_j = a]\!] \left(\boldsymbol{f}_{i_j}^\intercal \boldsymbol{\theta} - \widetilde{y}_{i_j,a}\right)^2 + \lambda \|\boldsymbol{\theta}\|^2 \tag{A.1}$$

The solution is given by $\widehat{\boldsymbol{\theta}}_{t,a} = (A_{t,a})^{-1}\boldsymbol{b}_{t,a}$, where $\boldsymbol{b}_{t,a} = X_{t,a}^\intercal \widetilde{\boldsymbol{y}}_{t,a}$ and $\widetilde{\boldsymbol{y}}_{t,a}$ is the (column) vector storing all observed and possibly imputed log-runtimes until t whenever a has been chosen. We follow the approach by Chu et al. [Chu+11] and analyze the deviation of the estimated log-runtime $\boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a}$ and the true log-runtime $\boldsymbol{x}_t^\intercal \boldsymbol{\theta}_a^*$ (according to our assumptions in Section 5.2). For this purpose, let $A_{t,a} = \lambda I_d + X_{t,a}^\intercal X_{t,a}$ be the regularized Gram matrix where $I_d \in \mathbb{R}^{d\times d}$ is the identity

matrix and $X_{t,a}$ denotes the design matrix at timestep t for algorithm a. Then, we have that

$$\begin{aligned}
\boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \boldsymbol{x}_t^\intercal \boldsymbol{\theta}_a^* &= \boldsymbol{x}_t^\intercal A_{t,a}^{-1} \boldsymbol{b}_{t,a} - \boldsymbol{x}_t^\intercal A_{t,a}^{-1} A_{t,a} \boldsymbol{\theta}_a^* \\
&= \boldsymbol{x}_t^\intercal A_{t,a}^{-1} X_{t,a}^\intercal \tilde{\boldsymbol{y}}_{t,a} - \boldsymbol{x}_t^\intercal A_{t,a}^{-1} \Big(\lambda I_d + X_{t,a}^\intercal X_{t,a}\Big) \boldsymbol{\theta}_a^* \\
&= \boldsymbol{x}_t^\intercal A_{t,a}^{-1} X_{t,a}^\intercal \Big(\tilde{\boldsymbol{y}}_{t,a} - X_{t,a} \boldsymbol{\theta}_a^*\Big) - \lambda \boldsymbol{x}_t^\intercal A_{t,a}^{-1} \boldsymbol{\theta}_a^* \\
&= \boldsymbol{z}_{t,a} (\tilde{\boldsymbol{y}}_{t,a} - X_{t,a} \boldsymbol{\theta}_a^*) - \lambda \boldsymbol{x}_t^\intercal A_{t,a}^{-1} \boldsymbol{\theta}_a^* \\
&=: (U1) - (U2),
\end{aligned} \tag{A.2}$$

where we abbreviated $\boldsymbol{z}_{t,a} = \boldsymbol{x}_t^\intercal A_{t,a}^{-1} X_{t,a}^\intercal$, which is a row vector with $N_a(t)$ components and $N_a(t)$ is the total number of times algorithm a has been chosen until timestep t. We can rewrite the first term $(U1)$ as

$$\sum_{1 \leqslant j \leqslant t : m_{i_j,a} \leqslant C} \boldsymbol{z}_{t,a}[j] \Big(\log(m(i_j, a)) - \boldsymbol{x}_{i_j}^\intercal \boldsymbol{\theta}_a^* \Big) + \sum_{1 \leqslant j \leqslant t : m_{i_j,a} > C} \boldsymbol{z}_{t,a}[j] \Big(\log(C) - \boldsymbol{x}_{i_j}^\intercal \boldsymbol{\theta}_a^* \Big),$$

i.e. we split it into the sum over all uncensored observations and the sum of all censored observations observed until timestep t.

This split is helpful as we will apply the Azuma-Hoeffding inequality [CL06, Lemma A.7] in the following in order to bound the first sum, i.e. the sum over all non-censored observations. To this end, let us define the martingale $X_0, \ldots, X_n$, where we assume for simplicity that $n = |\{j : 1 \leqslant j \leqslant t \wedge m_{i_j,a} \leqslant C\}|$, i.e. n is the number of summands of the left sum above. Let $j_1, \ldots, j_n$ be the timesteps for which $m_{i_j,a} \leqslant C$ holds. Then, let $X_w = \sum_{k=j_1}^{j_w} \boldsymbol{z}_{t,a}[k](\log(m(i_k, a) - \boldsymbol{x}_{i_k}^\top \boldsymbol{\theta}_a^*)$ for $1 \leqslant w \leqslant n$ and specifically $X_0 = 0$.

In order to see that $\{X_0, \ldots, X_n\}$ is indeed a martingale, we show that

$$\begin{aligned}
&\mathbb{E}\left[X_{j+1} | X_1, \ldots, X_j\right] = X_j \\
\Leftrightarrow\ &\mathbb{E}\left[X_{j+1} - X_j | X_1, \ldots, X_j\right] = 0 \\
\Leftrightarrow\ &\mathbb{E}\left[\boldsymbol{z}_{t,a}[j+1] \left(\log(m(i_{j+1}, a)) - \boldsymbol{x}_{i_{j+1}}^\intercal \boldsymbol{\theta}_a^*\right) | X_1, \ldots, X_j\right] = 0
\end{aligned} \tag{A.3}$$

for all $1 \leqslant j < n$.

Due to the linearity of expectation, we get that

$$\begin{aligned}
&\mathbb{E}\left[\boldsymbol{z}_{t,a}[j+1] \left(\log(m(i_{j+1}, a)) - \boldsymbol{x}_{i_{j+1}}^\intercal \boldsymbol{\theta}_a^*\right) | X_1, \ldots, X_j\right] \\
&= \boldsymbol{z}_{t,a}[j+1] \left(\mathbb{E}\left[\log(m(i_{j+1}, a))\right] - \boldsymbol{x}_{i_{j+1}}^\intercal \boldsymbol{\theta}_a^*\right) .
\end{aligned} \tag{A.4}$$

Now note that $\mathbb{E}\left[\log(m(i_j, a))\right] = \boldsymbol{x}_{i_j}^\intercal \boldsymbol{\theta}_a^*$ due to our assumptions made in Section 5.2 and hence

$$\begin{aligned}
&\mathbb{E}\left[\boldsymbol{z}_{t,a}[j+1]\left(\log(m(i_{j+1}, a)) - \boldsymbol{x}_{i_{j+1}}^\intercal \boldsymbol{\theta}_a^*\right) | X_1, \ldots, X_j\right] \\
&= \boldsymbol{z}_{t,a}[j+1]\left(\mathbb{E}\left[\log(m(i_{j+1}, a))\right] - \boldsymbol{x}_{i_{j+1}}^\intercal \boldsymbol{\theta}_a^*\right) \\
&= 0\,.
\end{aligned} \tag{A.5}$$

Correspondingly, $\{X_0, \ldots, X_n\}$ is indeed a martingale. In order to finally apply the Azuma-Hoeffding inequality, we still have to bound the difference for all $1 \leqslant j < n$

$$\begin{aligned}
|X_{j+1} - X_j| &= \left|\boldsymbol{z}_{t,a}[j+1]\left(\log(m(i_{j+1}, a)) - \boldsymbol{x}_{i_{j+1}}^\intercal \boldsymbol{\theta}_a^*\right)\right| \\
&\leqslant \boldsymbol{z}_{t,a}[j+1]\log(C)\,,
\end{aligned} \tag{A.6}$$

where we assume without loss of generality for the last step that all runtimes are rescaled such that the 1 instead of 0 is the minimum.

Then, by applying the Azuma-Hoeffding inequality we get for any $\tilde{\alpha} > 0$ that

$$\begin{aligned}
&\mathbb{P}\left(\left|\sum_{1 \leqslant j \leqslant t : m_{i_j,a} \leqslant C} \boldsymbol{z}_{t,a}[j]\big(\log(m(i_j, a)) - \boldsymbol{x}_{i_j}^\intercal \boldsymbol{\theta}_a^*\big)\right| > \tilde{\alpha}\, w_{t,a}(\boldsymbol{x}_t)\log(C)\right) \\
&\leqslant 2\exp\left(-\frac{2\,\tilde{\alpha}^2\, w_{t,a}^2(\boldsymbol{x}_t)}{\|\boldsymbol{z}_{t,a}\|^2}\right),
\end{aligned}$$

where $w_{t,a}(\boldsymbol{x}_t) = \|\boldsymbol{x}_t\|_{A_{t,a}}$ as in Section 5.3.

Note that

$$\begin{aligned}
\|\boldsymbol{z}_{t,a}\|^2 &= \boldsymbol{x}_t^\intercal A_{t,a}^{-1} X_{t,a}^T X_{t,a} A_{t,a}^{-1} \boldsymbol{x}_t \\
&\leqslant \boldsymbol{x}_t^\intercal A_{t,a}^{-1}(\lambda I_d + X_{t,a}^T X_{t,a}) A_{t,a}^{-1} \boldsymbol{x}_t \\
&= \boldsymbol{x}_t^\intercal A_{t,a}^{-1} \boldsymbol{x}_t \\
&= w_{t,a}^2(\boldsymbol{x}_t),
\end{aligned} \tag{A.7}$$

since $\lambda A_{t,a}^{-1} I_d A_{t,a}^{-1}$ is semi-positive definite. Thus, by choosing $\tilde{\alpha}$ appropriately the latter probability is small. In particular, we obtain that

$$\left| \sum_{1 \leqslant j \leqslant t : m_{i_j,a} \leqslant C} \boldsymbol{z}_{t,a}[j] \big(\log(m(i_j, a)) - \boldsymbol{x}_{i_j}^\intercal \boldsymbol{\theta}_a^* \big) \right| \leqslant \tilde{\alpha}\, w_{t,a}(\boldsymbol{x}_t) \log(C) \tag{A.8}$$

holds with high probability by choosing $\tilde{\alpha} > 0$ appropriately.

Now, it remains to bound the (absolute values of the) censored sum. Using the Cauchy-Schwarz inequality we can infer

$$\begin{aligned} \left| \sum_{1 \leqslant j \leqslant t : m_{i_j,a} > C} \boldsymbol{z}_{t,a}[j] \big(\log(C) - \boldsymbol{x}_{i_j}^\intercal \boldsymbol{\theta}_a^* \big) \right| &\leqslant \|\boldsymbol{z}_{t,a}\| \left\| \sum_{1 \leqslant j \leqslant t : m_{i_j,a} > C} \big(\log(C) - \boldsymbol{x}_{i_j}^\intercal \boldsymbol{\theta}_a^* \big) \right\| \\ &\overset{(A.7)}{\leqslant} w_{t,a}(\boldsymbol{x}_t) \left\| \sum_{1 \leqslant j \leqslant t : m_{i_j,a} > C} \big(\log(C) - \boldsymbol{x}_{i_j}^\intercal \boldsymbol{\theta}_a^* \big) \right\| \\ &\leqslant 2\, w_{t,a}(\boldsymbol{x}_t) \sqrt{N_{a,t}^{(C)}}\, \log(C), \end{aligned} \tag{A.9}$$

where we used for the last inequality that $\boldsymbol{x}_{i_j}^\intercal \boldsymbol{\theta}_a^* = \boldsymbol{f}_{i_j}^\intercal \boldsymbol{\theta}_a^* \leqslant \log(C)$ holds by our assumptions made in Section 5.2. Finally, the absolute value of the term $(U2)$ can be bounded as follows

$$\begin{aligned} |(U2)| &= \left| \lambda \boldsymbol{\theta}_a^* \boldsymbol{x}_t^\intercal A_{t,a}^{-1} \right| \\ &\leqslant \lambda \|\boldsymbol{\theta}_a^*\| \|\boldsymbol{x}_t^\intercal A_{t,a}^{-1}\| \\ &\overset{(A.7)}{\leqslant} \lambda \|\boldsymbol{\theta}_a^*\| w_{t,a}(\boldsymbol{x}_t). \end{aligned} \tag{A.10}$$

Combining (A.2)–(A.10), we have with high probability (depending on the choice of $\tilde{\alpha}$) that

$$\begin{aligned} |\boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \boldsymbol{x}_t^\intercal \boldsymbol{\theta}_a^*| &\leqslant |(U1)| + |(U2)| \\ &\leqslant \left(\tilde{\alpha} w_{t,a}(\boldsymbol{x}_t) \log(C) + 2 w_{t,a}(\boldsymbol{x}_t) \sqrt{N_{a,t}^{(C)}} \log(C) \right) + (\lambda \|\boldsymbol{\theta}_a^*\| w_{t,a}(\boldsymbol{x}_t)) \\ &= w_{t,a}(\boldsymbol{x}_t) \left(\lambda \|\boldsymbol{\theta}_a^*\| + \log(C) \left(\tilde{\alpha} + 2\sqrt{N_{a,t}^{(C)}} \right) \right). \end{aligned}$$

Thus, with

$$w_{t,a}^{(bc)}(\boldsymbol{x}_t) = \left(1 + 2\log(C) \left(1 + \sqrt{N_{a,t}^{(C)}} \right) \right) w_{t,a}(\boldsymbol{x}_t) \ , \tag{A.11}$$

and for some appropriate $\alpha > 0$ (which essentially hides the regularization term, i.e. $\lambda\|\boldsymbol{\theta}_a^*\|$), we have with high probability

$$|\boldsymbol{x}_t^\intercal \widehat{\boldsymbol{\theta}}_{t,a} - \boldsymbol{x}_t^\intercal \boldsymbol{\theta}_a^*| \leqslant \alpha w_{t,a}^{(bc)}(\boldsymbol{x}_t).$$

Bias issue. Note that from the definition of $w_{t,a}^{(bc)}(\boldsymbol{x}_t)$ we can see the bias issue of $\widehat{\boldsymbol{\theta}}_{t,a}$ due to the imputation employed. As well-known in the bandit community, the term $w_{t,a}(\boldsymbol{x}_t)$ is asymptotically tending against zero with the rate $\approx \sqrt{1/N_a(t)}$ [LS20, Chapter 19]. However, $w_{t,a}^{(bc)}(\boldsymbol{x}_t)$ does not tend to zero asymptotically for $t \to \infty$, if $\sqrt{N_{a,t}^{(C)}/N_a(t)} \to C'$ for some constant $C' > 0$. The latter condition, in turn, seems to be satisfied if for any t it holds that $\mathbb{P}(m_{i_t,a} > C) > \epsilon > 0$, i.e. the probability of observing a censored runtime does not vanish in the course of time. In that case both $N_{a,t}^{(C)}$ and $N_a(t)$ will continuously grow and thus $w_{t,a}^{(bc)}(\boldsymbol{x}_t)$ cannot tend to 0.

A.4.2 Deriving the Refined Expected Loss Representation

In this section, we provide the details for showing Equation 5.13. For this purpose, we need the following lemma showing the explicit form of the conditional expectation of a log-normal distribution under a certain cutoff.

Lemma 1. *Let $Y \sim \mathrm{LN}(\mu, \sigma^2)$, i.e. a log-normally distributed random variable with parameters $\mu \in \mathbb{R}$ and $\sigma > 0$. Then, for any $C > 0$ it holds that*

$$\mathbb{E}\left[Y | Y \leqslant C\right] = \exp(\mu + \sigma^2/2) \cdot \frac{\Phi_{0,1}\left(\frac{\log(C) - \mu - \sigma^2}{\sigma}\right)}{\Phi_{0,1}\left(\frac{\log(C) - \mu}{\sigma}\right)}, \tag{A.12}$$

where $\Phi_{0,1}(\cdot)$ is the cumulative distribution function of a standard normal distribution.

Proof. The density function of Y is given by

$$f(x) = \frac{\exp\left(\frac{-(\log(x)-\mu)^2}{2\sigma^2}\right)}{x\sigma\sqrt{2\pi}}, \quad x > 0.$$

Thus, $f(x) = \frac{\phi_{\mu,\sigma}(\log(x))}{x}$, where $\phi_{\mu,\sigma}(\cdot)$ is the density function of a normal distribution with mean μ and standard deviation σ. Next, note that the density function

of Y conditioned on $Y \leqslant C$ is $f(x|x \leqslant C) = \frac{f(x)}{F(C)}$, where $F(\cdot)$ is the cumulative distribution function of Y and given by $F(x) = \Phi_{0,1}\left(\frac{\log(x)-\mu}{\sigma}\right)$ for any $x \in \mathbb{R}$. With this,

$$\begin{aligned}
\mathbb{E}\left[Y|Y \leqslant C\right] &= \int_0^C x f(x|x \leqslant C)\,\mathrm{d}x \\
&= \frac{1}{\Phi_{0,1}\left(\frac{\log(C)-\mu}{\sigma}\right)} \int_0^C \phi_{\mu,\sigma}(\log(x))\,\mathrm{d}x \\
&= \frac{\exp(\mu + \sigma^2/2)}{\Phi_{0,1}\left(\frac{\log(C)-\mu}{\sigma}\right)} \int_{-\infty}^{\frac{\log(C)-\mu}{\sigma}} \phi_{0,1}(z-\sigma)\,\mathrm{d}z \\
&= \exp(\mu + \sigma^2/2)\frac{\Phi_{0,1}\left(\frac{\log(C)-\mu-\sigma^2}{\sigma}\right)}{\Phi_{0,1}\left(\frac{\log(C)-\mu}{\sigma}\right)}.
\end{aligned}$$

Here, we used for the third equality the substitution $z = \frac{\log(x)-\mu}{\sigma}$, so that $\exp(\sigma z + \mu)\sigma\,\mathrm{d}z = \mathrm{d}x$ and

$$\exp\left(-\frac{(z-\sigma)^2}{2}\right)\exp\left(\frac{\sigma^2}{2}\right) = \exp\left(-\frac{z^2}{2}\right)\exp(\sigma z).$$

□

Recalling the modeling assumption on the runtimes made in Equation 5.4 as well as the assumption that the error term is log-normally distributed, we obtain that $m_{i_t,a} \sim \mathrm{LN}(\boldsymbol{f}_{i_t}^\intercal \boldsymbol{\theta}_a^*, \sigma^2)$ as multiplying a log-normal by a constant results in obtaining

a log-normal again. Using Lemma 1 and that $C_{i_t,a}^{(1)} = {}^{(\log(C)-\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^*-\sigma^2)}/_{\sigma}$, $C_{i_t,a}^{(2)} = {}^{(\log(C)-\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^*)}/_{\sigma}$, we can derive Equation 5.13 as follows:

$$
\begin{aligned}
\mathbb{E}\left[l_{t,a}|\boldsymbol{f}_{i_t}\right] &= \mathbb{E}\left[l_{t,a}|m_{i_t,a} \leqslant C, \boldsymbol{f}_{i_t}\right] \cdot \mathbb{P}(m_{i_t,a} \leqslant C|\boldsymbol{f}_{i_t}) \\
&\quad + \mathbb{E}\left[l_{t,a}|m_{i_t,a} > C, \boldsymbol{f}_{i_t}\right] \cdot \mathbb{P}(m_{i_t,a} > C|\boldsymbol{f}_{i_t}) \\
&\overset{(A.12)}{=} \exp\left(\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^* + \frac{\sigma^2}{2}\right) \cdot \left(\frac{\Phi_{0,1}(C_{i_t,a}^{(1)})}{\Phi_{0,1}(C_{i_t,a}^{(2)})}\right) \cdot \mathbb{P}(m_{i_t,a} \leqslant C|\boldsymbol{f}_{i_t}) \\
&\quad + \mathcal{P}(C) \cdot \mathbb{P}(m_{i_t,a} > C|\boldsymbol{f}_{i_t}) \\
&= \exp\left(\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^* + \frac{\sigma^2}{2}\right) \cdot \left(\frac{\Phi_{0,1}(C_{i_t,a}^{(1)})}{\Phi_{0,1}(C_{i_t,a}^{(2)})}\right) \\
&\quad + \mathbb{P}(m_{i_t,a} > C|\boldsymbol{f}_{i_t}) \cdot \left(\mathcal{P}(C) - \exp\left(\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^* + \frac{\sigma^2}{2}\right) \cdot \frac{\Phi_{0,1}(C_{i_t,a}^{(1)})}{\Phi_{0,1}(C_{i_t,a}^{(2)})}\right) \\
&= E_C(\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^*, \sigma) + \left(1 - \Phi_{\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^*,\sigma}(\log(C))\right) \cdot \left(\mathcal{P}(C) - E_C(\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^*, \sigma)\right),
\end{aligned}
$$

where $E_C(\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^*, \sigma) = \exp(\boldsymbol{f}_{i_t}^\intercal\boldsymbol{\theta}_a^* + {}^{\sigma^2}/_2) \cdot \frac{\Phi_{0,1}(C_{i_t,a}^{(1)})}{\Phi_{0,1}(C_{i_t,a}^{(2)})}$.

A.4.3 Pseudocode and Space-Complexity Details

In this section, we provide details on the pseudocodes in Section 5.3 and Section 5.4 and some of the solutions to equations used there.

Note that it is straightforward to see that the solution of Equation 5.6 (a standard ridge regression problem) is $\widehat{\boldsymbol{\theta}}_{t,a} = (A_{t,a})^{-1}\boldsymbol{b}_{t,a}$, where $\boldsymbol{b}_{t,a} = X_{t,a}^\intercal\boldsymbol{y}_{t,a}$ and $\boldsymbol{y}_{t,a}$ is the (column) vector storing all observed non-censored log-runtimes until t whenever a has been chosen.

Similarly, the solution of Equation 5.9 is $\widehat{\boldsymbol{\theta}}_{t,a} = (A_{t,a})^{-1}\boldsymbol{b}_{t,a}$, where $\boldsymbol{b}_{t,a} = X_{t,a}^\intercal\tilde{\boldsymbol{y}}_{t,a}$ and $\tilde{\boldsymbol{y}}_{t,a}$ is the (column) vector storing all observed and possibly imputed log-runtimes until t whenever a has been chosen.

Further, both $A_{t,a}$ and $\boldsymbol{b}_{t,a}$ can be updated in an iterative fashion without actually storing all seen samples: If algorithm a is chosen at timestep t, then

$$
A_{t+1,a} = \begin{cases} A_{t,a} + \boldsymbol{f}_{i_t}\boldsymbol{f}_{i_t}^\intercal, & \text{for (5.9)}, \\ A_{t,a} + [\![m(i_t, a_t) \leqslant C]\!]\boldsymbol{f}_{i_t}\boldsymbol{f}_{i_t}^\intercal, & \text{for (5.6)}. \end{cases} \tag{A.13}
$$

$$\boldsymbol{b}_{t+1,a} = \begin{cases} \boldsymbol{b}_{t,a} + \tilde{y}_{i_t,a}\boldsymbol{f}_{i_t}, & \text{for (5.9)}, \\ \boldsymbol{b}_{t,a} + [\![m(i_t,a_t) \leqslant C]\!]y_{i_t,a}\boldsymbol{f}_{i_t}, & \text{for (5.6)}. \end{cases} \tag{A.14}$$

Correspondingly, our algorithms feature indeed a space-complexity independent of the time horizon.

As the updates of the matrices $A_{t+1,a}$ are via a rank-one update, one can use the well-known Sherman-Morrison formula to compute their inverse in a sequential manner as well (similarly for $A_{t+1,a}$ based on (5.6)):

$$(A_{t+1,a})^{-1} = (A_{t,a} + \boldsymbol{f}_{i_t}\boldsymbol{f}_{i_t}^\intercal)^{-1} = A_{t,a}^{-1} - \frac{A_{t,a}^{-1}\boldsymbol{f}_{i_t}\boldsymbol{f}_{i_t}^\intercal A_{t,a}^{-1}}{1 + \boldsymbol{f}_{i_t}^\intercal A_{t,a}^{-1}\boldsymbol{f}_{i_t}}. \tag{A.15}$$

A.5 Details on the Experimental Evaluation of Chapter 6

All code, including detailed documentation of the experiments and execution instructions, is available at GitHub[19].

A.5.1 Hardware

All experiments were run on machines featuring Intel Xeon E5-2695v4@2.1GHz CPUs with 16 cores and 64GB RAM.

A.5.2 Software

All experiments were implemented in Python 3, although some of the code for generating result tables was written in Java. A full list of used Python packages can be found at [20] and [21].

[19] https://github.com/alexandertornede/as_on_a_meta_level

[20] https://github.com/alexandertornede/as_on_a_meta_level/blob/main/meta_learning/python/anaconda/meta_as.yml

[21] https://github.com/alexandertornede/as_on_a_meta_level/blob/main/ensemble_learning/anaconda/ensemble_environment.yml

The most important packages used are scikit-learn[15], scikit-survival[12] and scipy[13].

A.5.3 Hyperparameter Settings

- all base algorithm selectors
  - settings as in Section A.2
- Voting
  - None
- Bagging
  - num base learners: 10
- Stacking
  - threshold (variance threshold feature selection variant): 0.16
  - k (select k best feature selection variant): number of algorithms of scenario
- Boosting
  - num iterations: 20

Tab. A.1: Average PAR10 scores (averaged over 10 seeds) and the corresponding standard deviation of all discussed approach variants and the Degroote approach.

Scenario \ Approach	bj_thompson_rev	bj_thompson	thompson_rev	thompson	bclinucb_rev	bclinucb
ASP-POTASSCO	929.45 ± 86.72	949.38 ± 62.38	**902.64** ± 78.43	916.28 ± 72.73	1318.15 ± 2.11	1473.45 ± 234.12
BNSL-2016	9656.86 ± 318.43	9638.04 ± 378.05	9467.01 ± 252.52	**9361.41** ± 262.75	11211.34 ± 4566.34	40601.32 ± 1015.89
CPMP-2015	7818.47 ± 1187.55	8241.01 ± 1164.85	8158.72 ± 1268.83	8499.38 ± 2059.18	7945.31 ± 41.97	12656.36 ± 489.88
CSP-2010	8138.63 ± 820.90	8295.76 ± 699.43	7892.67 ± 692.83	8103.96 ± 887.50	**7208.90** ± 11.16	9507.53 ± 2509.18
CSP-MZN-2013	8291.83 ± 589.63	8207.06 ± 532.70	8171.21 ± 594.49	8472.17 ± 760.25	12588.22 ± 2.44	14920.51 ± 660.01
CSP-Minizinc-Time-2016	**4741.99** ± 505.13	4811.54 ± 409.79	4759.50 ± 306.03	4942.91 ± 326.89	6544.69 ± 3.01	7346.72 ± 461.05
GRAPHS-2015	4.11e+07 ± 3.71e+06	4.13e+07 ± 4.39e+06	4.21e+07 ± 3.37e+06	3.97e+07 ± 4.76e+06	2.69e+08 ± 85095.74	2.64e+08 ± 5.58e+06
MAXSAT-PMS-2016	2774.15 ± 218.67	2853.44 ± 210.21	2808.51 ± 218.55	2763.58 ± 134.14	**2570.24** ± 13.60	15016.71 ± 311.64
MAXSAT-WPMS-2016	6548.69 ± 183.77	6304.15 ± 166.98	6592.87 ± 210.25	6527.34 ± 213.44	**5925.93** ± 11.42	15946.25 ± 217.82
MAXSAT12-PMS	5373.99 ± 348.92	5347.39 ± 291.87	5408.40 ± 482.42	5324.88 ± 208.49	**4900.51** ± 15.28	11106.24 ± 301.39
MAXSAT15-PMS-INDU	3040.87 ± 196.70	3046.05 ± 128.34	**3032.08** ± 90.71	3080.24 ± 130.48	14308.04 ± 8.74	14874.85 ± 219.37
MIP-2016	7961.45 ± 765.53	8081.57 ± 845.74	8746.73 ± 1159.36	8776.59 ± 823.11	**7780.33** ± 191.05	25635.03 ± 7904.63
PROTEUS-2014	14223.14 ± 766.02	**13484.34** ± 541.83	14115.69 ± 768.16	13550.56 ± 426.67	23353.59 ± 5.61	23445.04 ± 249.12
QBF-2011	15253.81 ± 839.93	15708.25 ± 784.81	15178.86 ± 904.72	14902.82 ± 834.91	17510.56 ± 11.72	25687.33 ± 89.05
QBF-2014	**3552.30** ± 192.75	3629.40 ± 220.68	3679.96 ± 256.03	3599.79 ± 193.80	4266.31 ± 4.51	6172.04 ± 238.02
QBF-2016	4770.36 ± 595.50	5082.59 ± 718.71	5045.16 ± 848.59	4937.47 ± 710.69	**4694.97** ± 137.14	8019.58 ± 1461.05
SAT03-16_INDU	12128.48 ± 477.79	**11980.15** ± 193.67	12154.46 ± 221.01	12225.57 ± 501.47	15989.74 ± 0.28	15461.82 ± 677.27
SAT11-HAND	30436.08 ± 1196.65	30484.08 ± 1379.35	30085.51 ± 764.32	29547.31 ± 871.64	32054.23 ± 101.09	31976.16 ± 851.07
SAT11-INDU	17083.58 ± 490.45	17540.58 ± 530.82	17028.84 ± 479.15	17360.28 ± 1064.28	19830.34 ± 106.64	17922.09 ± 993.15
SAT11-RAND	19656.39 ± 3747.94	18061.78 ± 2770.70	19061.88 ± 2522.11	**16535.77** ± 2649.93	19934.84 ± 0.01	23859.51 ± 902.77
SAT12-ALL	5110.38 ± 221.26	**4720.22** ± 432.14	5132.48 ± 395.74	4812.94 ± 387.46	7932.67 ± 4.88	8362.10 ± 196.28
SAT12-HAND	7707.60 ± 219.43	7443.01 ± 180.51	7509.02 ± 199.39	**7309.74** ± 138.02	9346.81 ± 7.74	9655.25 ± 335.09
SAT12-INDU	6227.99 ± 714.29	4511.68 ± 76.33	4945.79 ± 228.37	**4428.49** ± 142.91	11592.25 ± 3.04	11234.57 ± 265.75
SAT12-RAND	5489.81 ± 649.90	4008.79 ± 206.59	4523.33 ± 170.56	4157.19 ± 277.10	**3921.38** ± 4.31	11060.69 ± 158.63
SAT15-INDU	7885.50 ± 583.23	**7700.27** ± 310.65	7856.08 ± 522.84	7879.61 ± 546.81	16513.64 ± 53.52	21473.55 ± 841.13
SAT18-EXP	24942.85 ± 847.88	25201.41 ± 681.42	**24906.56** ± 540.36	25015.28 ± 1031.38	25294.33 ± 62.04	31202.35 ± 2815.63
TSP-LION2015	**1160.50** ± 396.73	1226.11 ± 309.42	1411.06 ± 329.16	1213.43 ± 370.00	11309.59 ± 159.51	11406.27 ± 55.62
avgrank	3.296296	3.37037	3.222222	**2.851852**	7.111111	12.0

Scenario \ Approach	blinducb_rev	blinducb	rand_bclinucb_rev	rand_bclinucb	rand_blinducb_rev	rand_blinducb	degroote_EpsilonGreedy_LinearRegression
ASP-POTASSCO	1471.79 ± 43.06	1204.20 ± 161.11	1337.57 ± 41.27	1220.15 ± 58.60	1280.76 ± 24.99	1198.79 ± 135.63	1047.13 ± 46.50
BNSL-2016	30500.86 ± 852.64	17369.40 ± 1553.72	30858.52 ± 1294.03	22164.52 ± 835.62	23563.48 ± 907.10	18531.72 ± 2259.16	12510.26 ± 1291.03
CPMP-2015	11667.65 ± 369.54	11518.55 ± 1316.62	10809.31 ± 381.66	10195.18 ± 570.98	10439.34 ± 308.03	11350.81 ± 1338.76	**6991.97** ± 501.36
CSP-2010	10348.84 ± 197.67	9563.85 ± 1821.74	10539.88 ± 261.24	9485.35 ± 262.55	9847.48 ± 316.70	8796.21 ± 1534.82	7593.13 ± 208.94
CSP-MZN-2013	14646.81 ± 64.27	12113.14 ± 593.67	13269.14 ± 94.42	12556.67 ± 75.59	13551.04 ± 95.32	11595.72 ± 914.74	**8034.62** ± 113.78
CSP-Minizinc-Time-2016	7138.65 ± 283.28	5944.38 ± 481.06	6719.46 ± 529.02	6274.06 ± 308.19	6291.28 ± 431.44	5353.62 ± 364.20	5258.70 ± 406.91
GRAPHS-2015	1.11e+08 ± 1.64e+07	6.91e+07 ± 4.15e+07	1.79e+08 ± 7.28e+06	1.06e+08 ± 1.41e+06	7.85e+07 ± 1.71e+07	6.72e+07 ± 4.00e+07	**3.45e+07** ± 1.42e+06
MAXSAT-PMS-2016	15087.51 ± 399.80	5785.25 ± 1369.27	13591.52 ± 241.68	7546.35 ± 350.79	11887.90 ± 403.61	5405.64 ± 495.98	3279.54 ± 133.00
MAXSAT-WPMS-2016	15948.34 ± 203.76	12940.70 ± 727.41	13704.32 ± 520.48	10765.11 ± 255.39	13492.59 ± 174.10	12518.20 ± 585.24	6287.21 ± 541.69
MAXSAT12-PMS	11918.42 ± 279.90	9829.22 ± 2477.59	10174.11 ± 229.65	8159.01 ± 180.27	9661.09 ± 254.48	9580.82 ± 2684.49	5308.11 ± 129.30
MAXSAT15-PMS-INDU	15482.08 ± 299.25	12020.48 ± 1801.39	14631.09 ± 265.77	9411.16 ± 246.88	12936.66 ± 495.89	10416.87 ± 930.70	3867.70 ± 255.98
MIP-2016	20305.81 ± 1787.80	10280.18 ± 646.54	22681.31 ± 2433.54	19274.82 ± 2372.50	16238.36 ± 1625.48	10102.14 ± 783.52	10644.68 ± 3405.18
PROTEUS-2014	23330.42 ± 206.84	23496.49 ± 957.96	22057.74 ± 460.76	21813.84 ± 547.78	22380.67 ± 173.49	23508.44 ± 1081.76	15622.29 ± 784.60
QBF-2011	24467.88 ± 266.89	24105.19 ± 2408.37	20894.53 ± 210.99	19522.13 ± 357.25	21819.49 ± 613.82	24190.56 ± 1452.98	**13912.24** ± 356.69
QBF-2014	5669.88 ± 58.89	4951.40 ± 630.45	5292.36 ± 127.24	5003.91 ± 99.27	5213.18 ± 81.48	4836.72 ± 617.95	4116.15 ± 116.27
QBF-2016	7631.75 ± 323.55	6080.61 ± 828.17	7640.48 ± 455.05	6373.15 ± 177.29	6811.82 ± 209.17	6234.42 ± 943.66	5346.29 ± 210.05
SAT03-16_INDU	13682.55 ± 263.59	12836.59 ± 385.99	13421.09 ± 304.61	13330.58 ± 142.67	13421.35 ± 251.79	12671.99 ± 458.24	12754.50 ± 200.55
SAT11-HAND	30670.46 ± 525.38	33755.48 ± 470.48	30818.45 ± 1000.82	30840.36 ± 863.06	30519.78 ± 633.95	33151.03 ± 689.91	**29544.70** ± 952.78
SAT11-INDU	17488.94 ± 420.92	17407.75 ± 887.53	17874.06 ± 897.49	17659.81 ± 909.60	17404.10 ± 671.54	17792.97 ± 645.24	**17018.24** ± 647.90
SAT11-RAND	32256.56 ± 333.93	32812.38 ± 484.62	24421.64 ± 748.42	23194.11 ± 342.96	28780.03 ± 577.18	32382.10 ± 453.97	21008.77 ± 530.22
SAT12-ALL	8548.46 ± 145.06	8235.03 ± 393.14	7058.96 ± 104.20	6813.39 ± 72.52	7417.48 ± 76.56	8231.51 ± 319.87	5650.32 ± 214.36
SAT12-HAND	8395.15 ± 289.83	9944.53 ± 702.20	8397.26 ± 177.31	8106.83 ± 130.61	8164.12 ± 126.91	9560.68 ± 629.12	7634.24 ± 267.89
SAT12-INDU	10586.81 ± 417.60	11709.07 ± 47.37	9655.50 ± 675.98	7799.19 ± 1271.96	9159.07 ± 827.64	11568.96 ± 45.53	4755.52 ± 206.95
SAT12-RAND	11118.21 ± 91.24	8270.41 ± 249.25	10392.11 ± 68.33	8885.34 ± 132.17	10432.59 ± 97.18	8109.35 ± 140.40	5023.73 ± 174.68
SAT15-INDU	9691.58 ± 716.04	11203.29 ± 3070.49	9552.26 ± 907.94	9716.33 ± 706.23	9599.63 ± 676.35	10656.72 ± 3172.70	8220.22 ± 525.13
SAT18-EXP	26762.26 ± 760.54	29573.37 ± 2210.54	26212.93 ± 719.32	26124.45 ± 837.33	26361.79 ± 1018.39	30237.60 ± 1497.64	25272.35 ± 881.19
TSP-LION2015	4810.21 ± 212.49	9991.79 ± 294.01	8147.81 ± 256.02	5686.03 ± 197.03	4588.53 ± 191.98	9422.52 ± 655.46	1634.79 ± 112.29
avgrank	10.814815	9.333333	9.814815	7.814815	8.740741	8.666667	3.962963

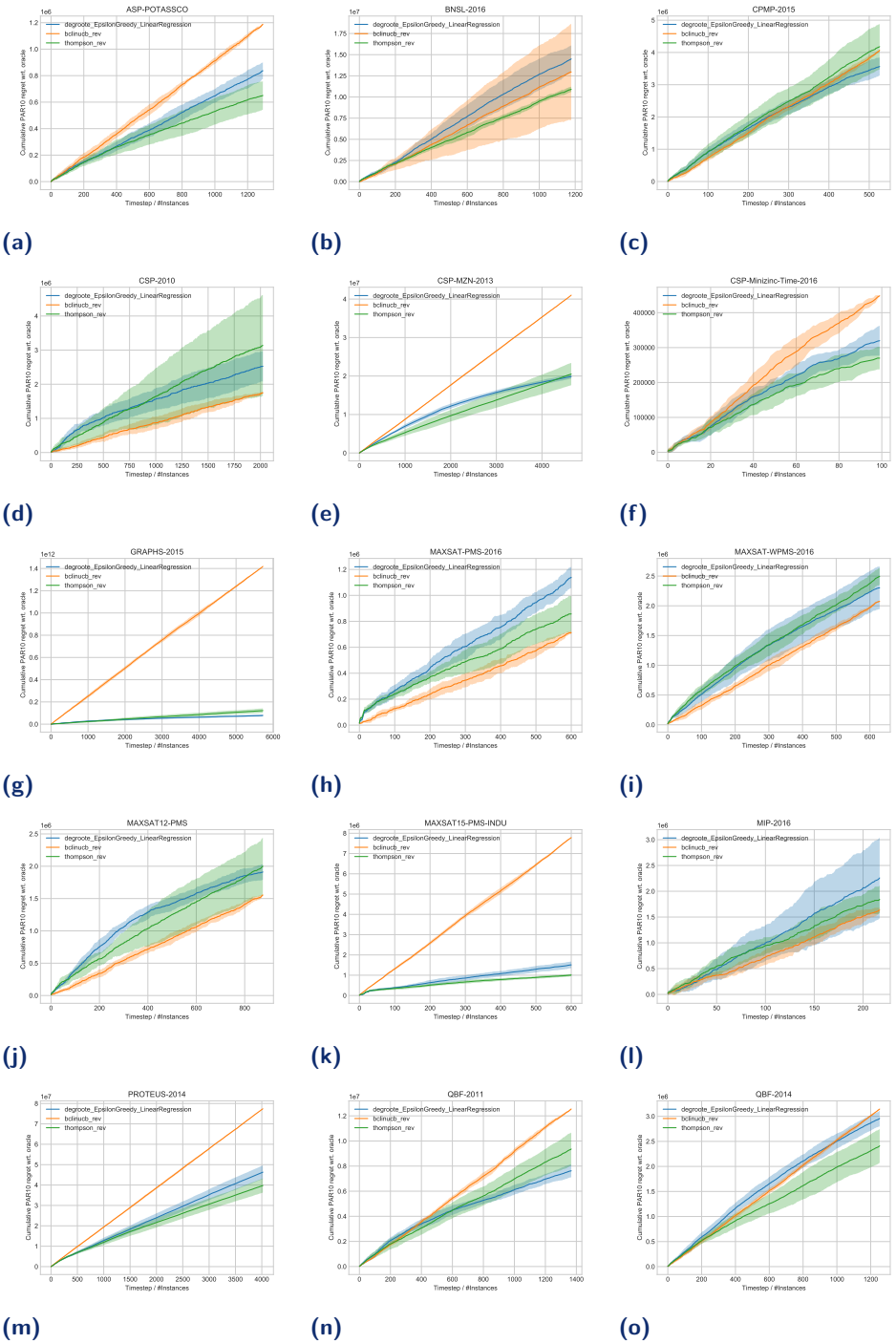

Fig. A.1: Cumulative PAR10 regret wrt. oracle.

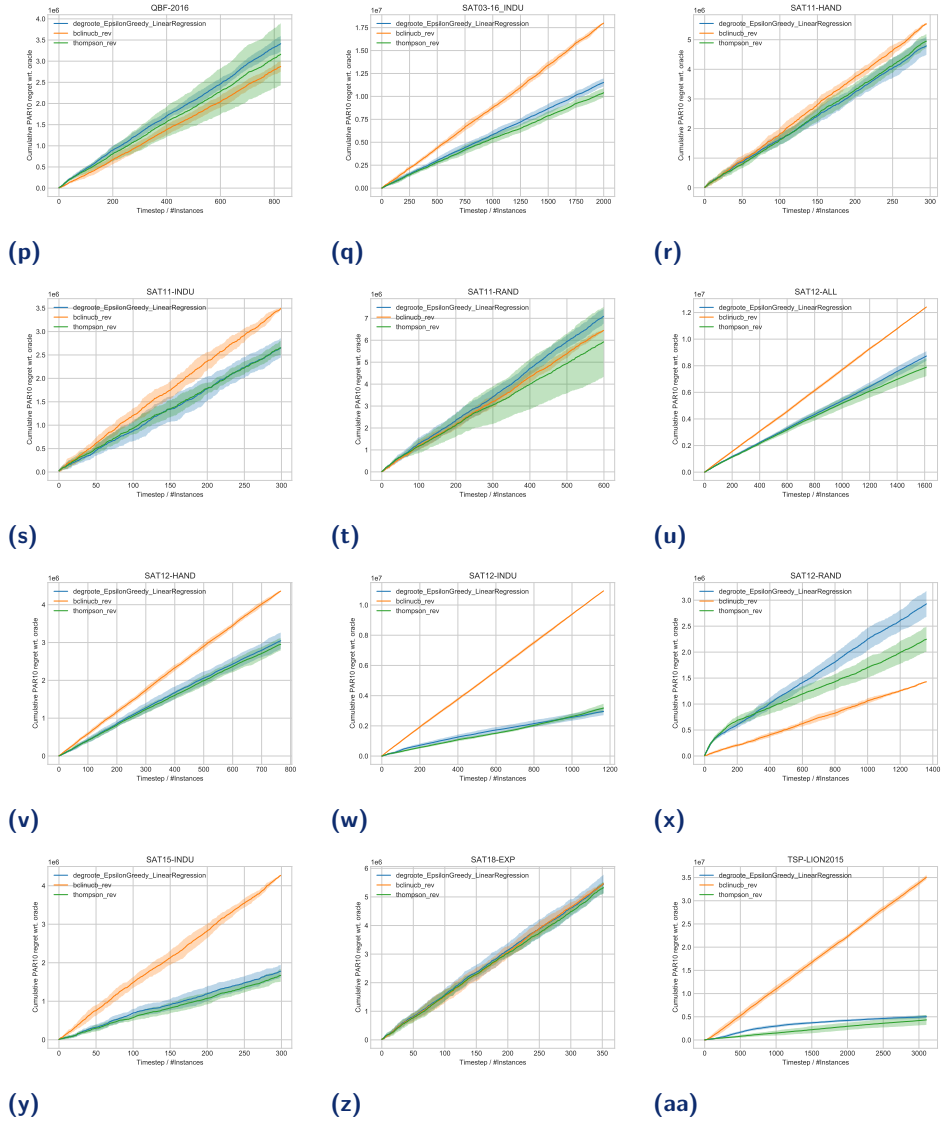

Fig. A.1: (Cont.) Cumulative PAR10 regret wrt. oracle.

Full List of my Publications

During my time working on this thesis, I was fortunate enough to publish more than just the papers, on which this thesis is actively based. In the following, a complete list of my 21 papers can be found:

F. Mohr, M. Wever, A. Tornede, and E. Hüllermeier. “From Automated to On-The-Fly Machine Learning”. In: *Proceedings of the INFORMATIK*. 2019.

A. Tornede, M. Wever, and E. Hüllermeier. “Algorithm Selection as Recommendation: From Collaborative Filtering to Dyad Ranking”. In: *29th Workshop Computational Intelligence*. **Young Author Award**, 2019.

M. Wever, F. Mohr, A. Tornede, and E. Hüllermeier. “Automating Multi-Label Classification Extending ML-Plan”. In: *ICML: Workshop on Automated Machine Learning*. 2019.

J. Hanselle, A. Tornede, M. Wever, and E. Hüllermeier. “Hybrid Ranking and Regression for Algorithm Selection”. In: *Proceedings of the 43rd German Conference on Artificial Intelligence (KI’2020)*. 2020.

A. Tornede, M. Wever, and E. Hüllermeier. “Extreme Algorithm Selection with Dyadic Feature Representation”. In: *Proceedings of the International Conference on Discovery Science (DS’20)*. 2020.

A. Tornede, M. Wever, and E. Hüllermeier. “Towards Meta-Algorithm Selection”. In: *NeurIPS: Workshop on Meta-Learning (MetaLearn’20)*. 2020.

A. Tornede, M. Wever, S. Werner, F. Mohr, and E. Hüllermeier. “Run2Survive: A Decision-theoretic Approach to Algorithm Selection based on Survival Analysis”. In: *Proceedings of the 12th Asian Conference on Machine Learning (ACML’20)*. 2020.

T. Tornede, A. Tornede, M. Wever, F. Mohr, and E. Hüllermeier. “AutoML for Predictive Maintenance: One Tool to RUL them All”. In: *ECML/PKDD: Workshop on IoT Streams for Data-Driven Predictive Maintenance*. 2020.

M. Wever, A. Tornede, F. Mohr, and E. Hüllermeier. “LiBRe: Label-Wise Selection of Base Learners in Binary Relevance for Multi-Label Classification”. In: *Proceedings of the 19th International Symposium on Intelligent Data Analysis (IDA’20)*. **Frontier Prize**, 2020.

J. Hanselle, A. Tornede, M. Wever, and E. Hüllermeier. “Algorithm Selection as Superset Learning: Constructing Algorithm Selectors from Imprecise Performance Data”. In: *Proceedings of the 25th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD’21)*. 2021.

E. Hüllermeier, F. Mohr, A. Tornede, and M. Wever. “Automated Machine Learning, Bounded Rationality, and Rational Metareasoning”. In: *ECML/PKDD: Workshop on Automating Data Science (ADS’21)*. 2021.

F. Mohr, M. Wever, A. Tornede, and E. Hüllermeier. “Predicting Machine Learning Pipeline Runtimes in the Context of Automated Machine Learning”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

T. Tornede, A. Tornede, J. Hanselle, M. Wever, F. Mohr, and E. Hüllermeier. “Towards Green Automated Machine Learning: Status Quo and Future Directions”. In: *arXiv:2111.05850* (2021).

T. Tornede, A. Tornede, M. Wever, and E. Hüllermeier. “Coevolution of Remaining Useful Lifetime Estimation Pipelines for Automated Predictive Maintenance”. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO’21)*. 2021.

M. Wever, A. Tornede, F. Mohr, and E. Hüllermeier. “AutoML for Multi-Label Classification: Overview and Empirical Evaluation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

L. Fehring, J. Hanselle, and A. Tornede. “HARRIS: Hybrid Ranking and Regression Forests for Algorithm Selection”. In: *NeurIPS: Workshop on Meta-Learning (MetaLearn’22)*. 2022.

K. Gevers, A. Tornede, M. Wever, V. Schöppner, and E. Hüllermeier. “A Comparison of Heuristic, Statistical, and Machine Learning Methods for Heated Tool Butt Welding of Two Different Materials”. In: *Welding in the World* (2022).

E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney. “A Survey of Methods for Automated Algorithm Configuration”. In: *Journal of Artificial Intelligence Research* (2022).

A. Tornede, V. Bengs, and E. Hüllermeier. “Machine Learning for Online Algorithm Selection under Censored Feedback”. In: *Proceedings of the Thirty-Sixth Conference on Artificial Intelligence (AAAI’22)*. 2022.

A. Tornede, L. Gehring, T. Tornede, M. Wever, and E. Hüllermeier. “Algorithm Selection on a Meta Level”. In: *Machine Learning* (2022).

T. Tornede, A. Tornede, L. Fehring, L. Gehring, H. Graf, J. Hanselle, F. Mohr, and M. Wever. “PyExperimenter: Easily Distribute Experiments and Track Results”. In: *Journal of Open Source Software* (2023).

Bibliography

[Aal78] O. Aalen. “Nonparametric Inference For a Family of Counting Processes”. In: *The Annals of Statistics* (1978) (cit. on p. 95).

[AG13] S. Agrawal and N. Goyal. “Thompson Sampling for Contextual Bandits with Linear Payoffs”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML’13)*. 2013 (cit. on p. 116).

[Ali10] A. Alin. “Multicollinearity”. In: *Wiley Interdisciplinary Reviews: Computational Statistics* (2010) (cit. on p. 51).

[ASH19] M. Alissa, K. Sim, and E. Hart. “Algorithm Selection Using Deep Learning Without Feature Extraction”. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO’19)*. 2019 (cit. on p. 56).

[ASH22] M. Alissa, K. Sim, and E. Hart. “Automated Algorithm Selection: From Feature-Based to Feature-Free Approaches”. In: *arXiv:2203.13392* (2022) (cit. on p. 56).

[Ama+15] R. Amadini, F. Biselli, M. Gabbrielli, T. Liu, and J. Mauro. “Feature Selection for SUNNY: A Study on the Algorithm Selection Library”. In: *Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI’15)*. 2015 (cit. on p. 57).

[AGM14] R. Amadini, M. Gabbrielli, and J. Mauro. “SUNNY: a Lazy Portfolio Approach for Constraint Solving”. In: *Theory and Practice of Logic Programming* (2014) (cit. on pp. 36, 101).

[Ame84] T. Amemiya. “Tobit Models: A Survey”. In: *Journal of Econometrics* (1984) (cit. on p. 106).

[Ana21] M. Anastacio. “Greybox Automated Algorithm Configuration”. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI’21) Doctoral Consortium Track*. 2021 (cit. on p. 174).

[Ans+15] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney. “Model-Based Genetic Algorithms for Algorithm Configuration”. In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI’15)*. 2015 (cit. on p. 86).

[Arm+06] W. Armstrong, P. Christen, E. McCreath, and A. P. Rendell. “Dynamic Algorithm Selection Using Reinforcement Learning”. In: *International Workshop on Integrating AI and Data Mining*. 2006 (cit. on p. 130).

[ACF02] P. Auer, N. Cesa-Bianchi, and P. Fischer. “Finite-Time Analysis of the Multiarmed Bandit Problem”. In: *Machine Learning* (2002) (cit. on p. 113).

[AutoML-FS] *AutoML Fall School*. https://sites.google.com/view/automl-fall-school-2022/home. Accessed: 2023-04-22 (cit. on p. 2).

[AutoML-Conf] *AutoML-Conf*. https://automl.cc/. Accessed: 2023-04-22 (cit. on p. 2).

[Bac17] F. Bach. "Breaking the Curse of Dimensionality With Convex Neural Networks". In: *Journal of Machine Learning Research* (2017) (cit. on p. 50).

[Ben+18] S. Bengio, K. Dembczynski, T. Joachims, M. Kloft, and M. Varma. "Extreme Classification (Dagstuhl Seminar 18291)". In: *Dagstuhl Reports* (2018) (cit. on pp. 61, 62, 65).

[Ben12] Y. Bengio. "Deep Learning of Representations for Unsupervised and Transfer Learning". In: *ICML: Workshop on Unsupervised and Transfer Learning*. 2012 (cit. on p. 54).

[BH20] V. Bengs and E. Hüllermeier. "Preselection bandits". In: *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. 2020 (cit. on p. 171).

[BH19] V. Bengs and E. Hüllermeier. "Preselection Bandits Under the Plackett-Luce Model". In: *arXiv:1907.06123* (2019) (cit. on p. 171).

[Bie+22] T. De Bie, L. De Raedt, J. Hernández-Orallo, H. H. Hoos, P. Smyth, and C. K. I. Williams. "Automating Data Science". In: *Communications of the ACM* (2022) (cit. on p. 24).

[Bie+20] A. Biedenkapp, H. F. Bozkurt, T. Eimer, F. Hutter, and M. Lindauer. "Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework". In: *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*. 2020 (cit. on pp. 130, 174).

[Bie+19] A. Biedenkapp, H. F. Bozkurt, F. Hutter, and M. Lindauer. "Towards White-box Benchmarks for Algorithm Control". In: *arXiv:1906.07644* (2019) (cit. on p. 130).

[Bie+17] A. Biedenkapp, M. Lindauer, K. Eggensperger, F. Hutter, C. Fawcett, and H. H. Hoos. "Efficient Parameter Importance Analysis via Ablation with Surrogates". In: *Proceedings of the Thirty-First Conference on Artificial Intelligence (AAAI'17)*. 2017 (cit. on pp. 98, 106).

[Bis+23] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A. Boulesteix, D. Deng, and M. Lindauer. "Hyperparameter Optimization: Foundations, Algorithms, Best Practices, and Open Challenges". In: *WIREs Data Mining and Knowledge Discovery* (2023) (cit. on p. 23).

[Bis+16] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren. "ASlib: A Benchmark Library for Algorithm Selection". In: *Artificial Intelligence* (2016) (cit. on pp. 13, 47, 53, 57, 90, 151).

[Bis+11] B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß. “Algorithm Selection Based on Exploratory Landscape Analysis and Cost-Sensitive Learning”. In: *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO’12)*. 2011 (cit. on p. 53).

[BMB13] M. R. Bonyadi, Z. Michalewicz, and L. Barone. “The Travelling Thief Problem: The First Step in the Transition From Theoretical Problems to Realistic Problems”. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC’13)*. 2013 (cit. on p. 162).

[Bor84] J. C. Borda. “Mémoire Sur Les élections au Scrutin”. In: *Histoire de l’Academie Royale des Sciences pour 1781* (1784) (cit. on p. 144).

[BT18] J. Bossek and H. Trautmann. “Multi-objective Performance Measurement: Alternatives to PAR10 and Expected Running Time”. In: *Proceedings of the International Conference on Learning and Intelligent Optimization (LION’18)*. 2018 (cit. on pp. 14, 47).

[Bra+08] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to Data Mining*. Springer Science & Business Media, 2008 (cit. on p. 139).

[Bra+22] P. Brazdil, J. N. van Rijn, C. Soares, and J. Vanschoren. “Automating Data Science”. In: *Metalearning*. Springer, 2022 (cit. on pp. 24, 51).

[Bre01] L. Breiman. “Random Forests”. In: *Machine Learning* (2001) (cit. on p. 156).

[Bre+84] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and Regression Trees*. 1984 (cit. on p. 94).

[Bre96] Leo Breiman. “Bagging Predictors”. In: *Machine Learning* (1996) (cit. on pp. 142, 146).

[Bre72] N. E. Breslow. “Contribution to Discussion of Paper by D. R. Cox”. In: *Journal of the Royal Statistical Society* (1972) (cit. on p. 109).

[BJ79] J. Buckley and I. James. “Linear Regression With Censored Data”. In: *Biometrika* (1979) (cit. on p. 121).

[Cao+07] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li. “Learning to Rank: From Pairwise Approach to Listwise Approach”. In: *Proceedings of the 24th International Conference on Machine Learning (ICML’07)*. 2007 (cit. on p. 69).

[CN06] R. Caruana and A. Niculescu-Mizil. “An Empirical Comparison of Supervised Learning Algorithms”. In: *Proceedings of the 23rd International Conference on Machine Learning (ICML’06)*. 2006 (cit. on p. 156).

[CB21] G. Casella and R. L. Berger. *Statistical Inference*. Cengage Learning, 2021 (cit. on p. 38).

[CL06] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006 (cit. on p. 186).

[CHH09] W. Cheng, J. Hühn, and E. Hüllermeier. “Decision Tree and Instance-Based Learning for Label Ranking”. In: *Proceedings of the 26th International Conference on Machine Learning (ICML’09)*. 2009 (cit. on p. 33).

[CHD10] W. Cheng, E. Hüllermeier, and K. J. Dembczynski. “Label Ranking Methods Based on the Plackett-Luce Model”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML’10)*. 2010 (cit. on pp. 33, 68, 69).

[Chu+11] W. Chu, L. Li, L. Reyzin, and R. E. Schapire. “Contextual Bandits With Linear Payoff Dunctions”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS’11)*. 2011 (cit. on pp. 113, 185).

[CS05] V. A. Cicirello and S. F. Smith. “The Max K-Armed Bandit: A New Model of Exploration Applied to Search Heuristic Selection”. In: *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI’05)*. 2005 (cit. on p. 130).

[CFR06] D. Coppersmith, L. Fleischer, and A. Rudra. “Ordering by Weighted Number of Wins Gives a Good Ranking for Weighted Tournaments”. In: *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA’06)*. 2006 (cit. on p. 145).

[COSEAL] *COSEAL*. `https://www.coseal.net/`. Accessed: 2023-04-22 (cit. on p. 2).

[Cox72] D. R. Cox. “Regression Models and Life Tables (With Discussion)”. In: *Journal of the Royal Statistical Society* (1972) (cit. on p. 109).

[CSC18] T. Cunha, C. Soares, and A. C. P. L. F. de Carvalho. “A Label Ranking Approach for Selecting Rankings of Collaborative Filtering Algorithms”. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC’18)*. 2018 (cit. on p. 33).

[dWB21] J. de Nobel, H. Wang, and T. Baeck. “Explorative Data Analysis of Time Series Based Algorithm Features of CMA-ES Variants”. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO’21)*. 2021 (cit. on pp. 86, 87).

[Deb14] K. Deb. “Multi-Objective Optimization”. In: *Search Methodologies*. Springer, 2014 (cit. on p. 47).

[Deg17] H. Degroote. “Online Algorithm Selection”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI’17)*. 2017 (cit. on p. 129).

[Deg+16] H. Degroote, B. Bischl, L. Kotthoff, and P. De Causmaecker. “Reinforcement Learning for Automatic Online Algorithm Selection - An Empirical Study”. In: *Proceedings of the 16th ITAT Conference Information Technologies*. 2016 (cit. on p. 129).

[Deg+18] H. Degroote, P. De Causmaecker, B. Bischl, and L. Kotthoff. "A Regression-Based Methodology for Online Algorithm Selection". In: *Proceedings of the Eleventh Annual Symposium on Combinatorial Search (SOCS'18)*. 2018 (cit. on pp. 126, 127, 129, 130, 183).

[DS21] N. Deshpande and N. Sharma. "Algorithm Selection Using Transfer Learning". In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'21)*. 2021 (cit. on p. 173).

[Die00] T. G. Dietterich. "Ensemble Methods in Machine Learning". In: *Proceedings of the First International Workshop on Multiple Classifier Systems (MCS'00)*. 2000 (cit. on pp. 18, 138, 141, 142).

[Dim+19] M. Dimakopoulou, Z. Zhou, S. Athey, and G. Imbens. "Balanced Linear Contextual Bandits". In: *Proceedings of the Thirty-Third Conference on Artificial Intelligence (AAAI'19)*. 2019 (cit. on p. 115).

[DBH18] F. K. Došilović, M. Brčić, and N. Hlupić. "Explainable Artificial Intelligence: A Survey". In: *Proceedings of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO'18)*. 2018 (cit. on p. 51).

[Dru97] H. Drucker. "Improving Regressors Using Boosting Techniques". In: *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*. 1997 (cit. on p. 148).

[DLH19] M. Du, N. Liu, and X. Hu. "Techniques for Interpretable Machine Learning". In: *Communications of the ACM* (2019) (cit. on p. 51).

[Dwo+01] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. "Rank Aggregation Methods for the Web". In: *Proceedings of the Tenth International World Wide Web Conference (WWW'01)*. 2001 (cit. on pp. 144, 145).

[Egg+20] K. Eggensperger, K. Haase, P. Müller, M. Lindauer, and F. Hutter. "Neural Model-based Optimization with Right-Censored Observations". In: *arXiv: 2009.13828* (2020) (cit. on p. 106).

[Egg+18] K. Eggensperger, M. Lindauer, H. H. Hoos, F. Hutter, and K. Leyton-Brown. "Efficient Benchmarking of Algorithm Configurators via Model-Based Surrogates". In: *Machine Learning* (2018) (cit. on pp. 91, 106).

[EVR09] B. Eksioglu, A.V. Vural, and A. Reisman. "The Vehicle Routing Problem: A Taxonomic Review". In: *Computers & Industrial Engineering* (2009) (cit. on p. 1).

[El +20] A. El Mesaoudi-Paul, D. Weiß, V. Bengs, E. Hüllermeier, and K. Tierney. "Pool-Based Realtime Algorithm Configuration: A Preselection Bandit Approach". In: *Proceedings of the International Conference on Learning and Intelligent Optimization (LION'20)*. 2020 (cit. on p. 130).

[EMH19] T. Elsken, J. H. Metzen, and F. Hutter. "Neural Architecture Search: A Survey". In: *Journal of Machine Learning Research* (2019) (cit. on p. 24).

[ED18] M. T. M. Emmerich and A. H. Deutz. “A Tutorial on Multiobjective Optimization: Fundamentals and Evolutionary Methods”. In: *Natural computing* (2018) (cit. on p. 47).

[Eri+20] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. “AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data”. In: *arXiv:2003.06505* (2020) (cit. on pp. 166, 167).

[Faw+14] C. Fawcett, M. Vallati, F. Hutter, J. Hoffmann, H. H. Hoos, and K. Leyton-Brown. “Improved Features for Runtime Prediction of Domain-Independent Planners”. In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS-14)*. 2014 (cit. on p. 57).

[FHT22] L. Fehring, J. Hanselle, and A. Tornede. “HARRIS: Hybrid Ranking and Regression Forests for Algorithm Selection”. In: *NeurIPS: Workshop on Meta-Learning (MetaLearn’22)*. 2022 (cit. on pp. 34, 172).

[Feu+22] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. “Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning”. In: *Journal of Machine Learning Research* (2022) (cit. on p. 167).

[FH18] M. Feurer and F. Hutter. “Towards Further Automation in AutoML”. In: *ICML: Workshop on Automated Machine Learning*. 2018 (cit. on pp. 41, 166).

[Feu+15] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. “Efficient and Robust Automated Machine Learning”. In: *Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NeurIPS’15)*. 2015 (cit. on pp. 62, 166).

[Fis69] P. C. Fishburn. *Utility-Theory for Decision Making*. Wiley, 1969 (cit. on p. 98).

[FMO15] T. Fitzgerald, Y. Malitsky, and B. O’Sullivan. “ReACTR: Realtime Algorithm Configuration Through Tournament Rankings”. In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI’15)*. 2015 (cit. on p. 130).

[Fit+14] T. Fitzgerald, Y. Malitsky, B. O’Sullivan, and K. Tierney. “ReACT: Real-Time Algorithm Configuration Through Tournaments”. In: *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SOCS’14)*. 2014 (cit. on p. 130).

[Fra+05] E. Frank, M. A. Hall, G. Holmes, R. Kirkby, and B. Pfahringer. “WEKA - A Machine Learning Workbench for Data Mining”. In: *The Data Mining and Knowledge Discovery Handbook*. 2005 (cit. on p. 72).

[Fra18] P. I. Frazier. “A Tutorial on Bayesian Optimization”. In: *arXiv:1807.02811* (2018) (cit. on p. 99).

[FLS04] D. Frossyniotis, A. Likas, and A. Stafylopatis. “A Clustering Method Based on Boosting”. In: *Pattern Recognition Letters* (2004) (cit. on pp. 157, 167).

[FSE18] N. Fusi, R. Sheth, and M. Elibol. “Probabilistic Matrix Factorization for Automated Machine Learning”. In: *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS’18)*. 2018 (cit. on p. 86).

[GL10] M. Gagliolo and C. Legrand. “Algorithm Survival Analysis”. In: *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, 2010 (cit. on pp. 105, 130).

[GS10] M. Gagliolo and J. Schmidhuber. “Algorithm Selection as a Bandit Problem with Unbounded Losses”. In: *Proceedings of the Fourth International Conference on Learning and Intelligent Optimization (LION’10)*. 2010 (cit. on p. 130).

[GS06] M. Gagliolo and J. Schmidhuber. “Learning Dynamic Algorithm Portfolios”. In: *Annals of Mathematics and Artificial Intelligence* (2006) (cit. on pp. 96, 105, 130).

[Gam12] J. Gama. “A Survey on Learning From Data Streams: Current and Future Trends”. In: *Progress in Artificial Intelligence* (2012) (cit. on p. 130).

[GO09] N. García-Pedrajas and D. Ortiz-Boyer. “Boosting K-Nearest Neighbor Classifier by Means of Input Space Projection”. In: *Expert Systems with Applications* (2009) (cit. on pp. 157, 167).

[Gen+10] I. P. Gent, C. Jefferson, L. Kotthoff, I. Miguel, N. C. A. Moore, P. Nightingale, and K. Petrie. “Learning When to Use Lazy Learning in Constraint Solving”. In: *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI’10)*. 2010 (cit. on pp. 29, 53).

[GSC97] C. P. Gomes, B. Selman, and N. Crato. “Heavy-Tailed Distributions in Combinatorial Search”. In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP’97)*. 1997 (cit. on pp. 45, 89, 91, 162).

[GLR22] D. R. Graham, K. Leyton-Brown, and T. Roughgarden. “Formalizing Preferences Over Runtime Distributions”. In: *arXiv:2205.13028* (2022) (cit. on p. 47).

[Gre05] W. H. Greene. “Censored Data and Truncated Distributions”. In: *NYU Working Paper* (2005) (cit. on pp. 115, 125).

[GM04] A. Guerri and M. Milano. “Learning Techniques for Automatic Algorithm Portfolio Selection”. In: *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI’04)*. 2004 (cit. on p. 29).

[GR17] R. Gupta and T. Roughgarden. “A PAC Approach to Application-Specific Algorithm Selection”. In: *SIAM Journal on Computing* (2017) (cit. on p. 131).

[GE03] I. Guyon and A. Elisseeff. “An Introduction to Variable and Feature Selection”. In: *Journal of Machine Learning Research* (2003) (cit. on pp. 57, 149).

[HW09] S. Haim and T. Walsh. “Restart Strategy Selection Using Machine Learning Techniques”. In: *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT’09)*. 2009 (cit. on p. 31).

[HE03] G. Hamerly and C. Elkan. “Learning the K in K-Means”. In: *Proceedings of the 16th International Conference on Advances in Neural Information Processing Systems (NeurIPS’03)*. 2003 (cit. on p. 36).

[Han+21] J. Hanselle, A. Tornede, M. Wever, and E. Hüllermeier. “Algorithm Selection as Superset Learning: Constructing Algorithm Selectors from Imprecise Performance Data”. In: *Proceedings of the 25th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD’21)*. 2021 (cit. on p. 106).

[Han+20] J. Hanselle, A. Tornede, M. Wever, and E. Hüllermeier. “Hybrid Ranking and Regression for Algorithm Selection”. In: *Proceedings of the 43rd German Conference on Artificial Intelligence (KI’2020)*. 2020 (cit. on pp. 34, 172).

[Hap+13] M. Happe, F. Meyer auf der Heide, P. Kling, M. Platzner, and C. Plessl. “On-the-fly Computing: A Novel Paradigm for Individualized IT Services”. In: *Proceedings of the 16th IEEE International Symposium on Real-Time Distributed Computing (ISORC’13)*. 2013 (cit. on p. 62).

[Has+09] T. Hastie, S. Rosset, J. Zhu, and H. Zou. “Multi-Class Adaboost”. In: *Statistics and its Interface* (2009) (cit. on p. 148).

[Hei+21] J. Heins, J. Bossek, J. Pohl, M. Seiler, H. Trautmann, and P. Kerschke. “On the Potential of Normalized TSP Features for Automated Algorithm Selection”. In: *Proceedings of the 16th ACM/SIGEVO Conference on Foundations of Genetic Algorithms (FOGA’21)*. 2021 (cit. on p. 57).

[HMS09] D. Hernández-Lobato, G. Martínez-Muñoz, and A. Suárez. “Statistical Instance-Based Pruning in Ensembles of Independent Classifiers”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2009) (cit. on p. 146).

[Hil+09] M. Hilario, A. Kalousis, P. Nguyen, and A. Woznica. “A Data Mining Ontology for Algorithm Selection and Meta-Mining”. In: *ECML/PKDD: Workshop on 3rd generation Data Mining*. 2009 (cit. on p. 86).

[Hoo+15] H. H. Hoos, R. Kaminski, M. Lindauer, and T. Schaub. “aspeed: Solver Scheduling via Answer Set Programming”. In: *Theory and Practice of Logic Programming* (2015) (cit. on p. 20).

[HW06] P.D. Hough and P. J. Williams. *Modern Machine Learning for Automatic Optimization Algorithm Selection*. Tech. rep. Sandia National Lab.(SNL-CA), Livermore, CA (United States), 2006 (cit. on pp. 29, 86).

[HC15] E. Hüllermeier and W. Cheng. “Superset Learning Based on Feneralized Loss Minimization”. In: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD’15)*. 2015 (cit. on p. 106).

[Hül+21] E. Hüllermeier, F. Mohr, A. Tornede, and M. Wever. “Automated Machine Learning, Bounded Rationality, and Rational Metareasoning”. In: *ECML/PKDD: Workshop on Automating Data Science (ADS’21)*. 2021 (cit. on pp. 165, 171).

[Hut+06] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. “Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms”. In: *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP’06)*. 2006 (cit. on p. 31).

[HHL11] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Bayesian Optimization With Censored Response Data”. In: *NIPS: Workshop on Bayesian Optimization, Sequential Experimental Design and Bandits*. 2011 (cit. on pp. 91, 106).

[HHL13] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Identifying Key Algorithm Parameters and Instance Features using Forward Selection”. In: *Proceedings of the Seventh International Conference on Learning and Intelligent Optimization (LION’13)*. 2013 (cit. on p. 57).

[HHL] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. In: *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION’11)* (cit. on pp. 67, 86).

[HKV19] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature, 2019 (cit. on pp. 2, 23, 62).

[Hut+14] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. “Algorithm Runtime Prediction: Methods & Evaluation”. In: *Artificial Intelligence* (2014) (cit. on pp. 31, 53, 106).

[22] *ICML: Workshop on Adaptive Experimental Design and Active Learning in the Real World (ReALML’22)*. 2022.

[Ish+08] H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. “Random Survival Forests”. In: *The Annals of Applied Statistics* (2008) (cit. on p. 94).

[Kad+] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. “Algorithm Selection and Scheduling”. In: *Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP’11)* (cit. on p. 20).

[Kad+10] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. “ISAC - Instance-Specific Algorithm Configuration”. In: *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI’10)*. 2010 (cit. on pp. 36, 101).

[KM58] E. L. Kaplan and P. Meier. “Nonparametric Estimation From Incomplete Observations”. In: *Journal of the American Statistical Association* (1958) (cit. on p. 121).

[Ker+19] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. “Automated Algorithm Selection: Survey and Perspectives”. In: *Evolutionary Computation* (2019) (cit. on pp. 1, 2, 14, 17, 28, 51, 53, 54, 138, 231).

[KT19] P. Kerschke and H. Trautmann. “Automated Algorithm Selection on Continuous Black-Box Problems by Combining Exploratory Landscape Analysis and Machine Learning”. In: *Evolutionary Computation* (2019) (cit. on p. 53).

[Khu+16] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. “Cognito: Automated feature engineering for supervised learning”. In: *Proceedings of Workshops of the 16th IEEE International Conference on Data Mining (ICDM’16)*. 2016 (cit. on p. 24).

[KW16] T. N. Kipf and M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *arXiv:1609.02907* (2016) (cit. on p. 86).

[KK10] D. G. Kleinbaum and M. Klein. *Survival Analysis*. Springer, 2010 (cit. on pp. 90, 92).

[KCF18] P. Kordík, J. Cerný, and T. Frýda. “Discovering Predictive Ensembles for Transfer Learning and Meta-Learning”. In: *Machine Learning* (2018) (cit. on p. 166).

[Kor08] Y. Koren. “Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model”. In: *The 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’08)*. 2008 (cit. on pp. 38, 41).

[Kot16] L. Kotthoff. “Algorithm Selection for Combinatorial Search Problems: A Survey”. In: *Data Mining and Constraint Programming*. Springer, 2016 (cit. on pp. 2, 28).

[Kot12] L. Kotthoff. “Hybrid Regression-Classification Models for Algorithm Selection”. In: *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI’12)*. 2012 (cit. on pp. 29, 166).

[Kot14] L. Kotthoff. “Ranking Algorithms by Performance”. In: *Proceedings of the Eighth International Conference on Learning and Intelligent Optimization (LION’14)*. 2014 (cit. on p. 33).

[Kot+15] L. Kotthoff, P. Kerschke, H. H. Hoos, and H. Trautmann. “Improving the State of the Art in Inexact TSP Solving Using Per-Instance Algorithm Selection”. In: *Proceedings of the Nineth International Conference on Learning and Intelligent Optimization (LION’15)*. 2015 (cit. on p. 57).

[KM11] C. Kroer and Y. Malitsky. “Feature Filtering for Instance-Specific Algorithm Configuration”. In: *Proceedings of the 23th IEEE International Conference on Tools with Artificial Intelligence (ICTAI’11)*. 2011 (cit. on p. 57).

[LL00] M. G. Lagoudakis and M. L. Littman. “Algorithm Selection Using Reinforcement Learning”. In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML’00)*. 2000 (cit. on p. 130).

[LF17] R. Laroche and R. Féraud. “Algorithm Selection of Off-Policy Reinforcement Learning Algorithm”. In: *arXiv:1701.08810* (2017) (cit. on p. 130).

[LS20] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020 (cit. on pp. 113, 114, 189).

[LO01] A. Lazarevic and Z. Obradovic. “Effective Pruning of Neural Network Classifier Ensembles”. In: *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN’01)*. 2001 (cit. on p. 146).

[LB95] Y. LeCun and Y. Bengio. *Convolutional Networks for Images, Speech, and Time Series*. 1995 (cit. on p. 54).

[LNS02] K. Leyton-Brown, E. Nudelman, and Y. Shoham. “Learning the Empirical Hardness of Optimization Problems: The Case of Combinatorial Auctions”. In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP’02*. 2002 (cit. on p. 31).

[LBH16] M. Lindauer, R. Bergdoll, and F. Hutter. “An Empirical Study of Per-Instance Algorithm Scheduling”. In: *Proceedings of the Tenth International Conference on Learning and Intelligent Optimization (LION’16)*. 2016 (cit. on p. 130).

[Lin+22] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. “SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization”. In: *Journal of Machine Learning Research* (2022) (cit. on pp. 67, 86).

[LH12] M. Lindauer and H. H. Hoos. “Quantifying Homogeneity of Instance Sets for Algorithm Configuration”. In: *Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION’12)*. 2012 (cit. on p. 84).

[Lin+15] M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub. “AutoFolio: An Automatically Configured Algorithm Selector”. In: *Journal of Artificial Intelligence Research* (2015) (cit. on pp. 41, 47, 99, 167).

[LRK19] M. Lindauer, J. N. van Rijn, and L. Kotthoff. “The Algorithm Selection Competitions 2015 and 2017”. In: *Artificial Intelligence* (2019) (cit. on pp. 2, 17, 138, 166).

[LOW20] A. Lissovoi, P. S. Oliveto, and J. A. Warwicker. “Simple Hyper-Heuristics Control the Neighbourhood Size of Randomised Local Search Optimally for LeadingOnes*”. In: *Evolutionary Computation* (2020) (cit. on p. 130).

[Lor+16] A. Loreggia, Y. Malitsky, H. Samulowitz, and V. Saraswat. “Deep Learning for Algorithm Portfolios”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI’16)*. 2016 (cit. on pp. 55, 56).

[LCG08] A. C. Lorena, A. C. P. L. F. De Carvalho, and J. Gama. “A Review on the Combination of Binary Classifiers in Multiclass Problems”. In: *Artificial Intelligence Review* (2008) (cit. on p. 29).

[Mah+19] M. Mahdavi, F. Neutatz, L. Visengeriyeva, and Z. Abedjan. “Towards Automated Data Cleaning Workflows”. In: *Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen" (LWDA'19)*. 2019 (cit. on p. 23).

[MO14] Y. Malitsky and B. O'Sullivan. “Latent Features for Algorithm Selection”. In: *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SOCS'14)*. 2014 (cit. on pp. 40, 41, 173).

[Mal+13] Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. “Algorithm Portfolios Based on Cost-Sensitive Hierarchical Clustering”. In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*. 2013 (cit. on p. 36).

[Mal+11] Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. “Non-Model-Based Algorithm Portfolios for SAT”. In: *Proceedings of the Fourteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'11)*. 2011 (cit. on p. 29).

[Mal57] C. L. Mallows. “Non-Null Ranking Models”. In: *Biometrika* (1957) (cit. on p. 68).

[Mal+17] B. Malone, K. Kangas, M. Järvisalo, M. Koivisto, and P. Myllymäki. “asasl: Algorithm Selection with auto-sklearn”. In: *Open Algorithm Selection Challenge 2017*. 2017 (cit. on p. 166).

[Mar08] J. Marques-Silva. “Practical Applications of Boolean Satisfiability”. In: *9th International Workshop on Discrete Event Systems*. 2008 (cit. on p. 1).

[Mer+13] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph. “Exploratory Landscape Analysis”. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'13)*. 2013 (cit. on pp. 53, 174).

[MTL] *Meta-Learning Workshop*. `https://meta-learn.github.io/`. Accessed: 2023-04-22 (cit. on p. 2).

[MH82] R. Miller and J. Halpern. “Regression with Censored Data”. In: *Biometrika* (1982) (cit. on p. 121).

[MS13] M. Misir and M. Sebag. *Algorithm Selection as a Collaborative Filtering Problem*. Tech. rep. 2013 (cit. on pp. 40, 41).

[MS17] M. Misir and M. Sebag. “ALORS: An Algorithm Recommender System”. In: *Artificial Intelligence* (2017) (cit. on pp. 40, 41, 76).

[MRL22] A. Mohan, T. Ruhkopf, and M. Lindauer. “Towards Meta-learned Algorithm Selection using Implicit Fidelity Information”. In: *ICML: Workshop on Adaptive Experimental Design and Active Learning in the Real World (ReALML'22)*. 2022 (cit. on p. 174).

[Mv22] F. Mohr and J. N. van Rijn. “Learning Curves for Decision Making in Supervised Machine Learning–A Survey”. In: *arXiv:2201.12150* (2022) (cit. on p. 174).

[MWH18] F. Mohr, M. Wever, and E. Hüllermeier. “ML-Plan: Automated Machine Learning Via Hierarchical Planning”. In: *Machine Learning* (2018) (cit. on p. 62).

[Moh+19] F. Mohr, M. Wever, A. Tornede, and E. Hüllermeier. “From Automated to On-The-Fly Machine Learning”. In: *Proceedings of the INFORMATIK*. 2019 (cit. on pp. 62, 71).

[Moo+22] J. Moosbauer, G. Casalicchio, M. Lindauer, and B. Bischl. “Enhancing Explainability of Hyperparameter Optimization via Bayesian Algorithm Execution”. In: *arXiv:2206.05447* (2022) (cit. on p. 24).

[Nel72] W. Nelson. “Theory and Applications of Hazard Plotting for Censored Failure Data”. In: *Technometrics* (1972) (cit. on p. 95).

[Nud+04a] E. Nudelman, K. Leyton-Brown, A. Devkar, Y. Shoham, and H. H. Hoos. “SATzilla: An Algorithm Portfolio for SAT”. In: *SAT Competition 2004* (2004) (cit. on p. 31).

[Nud+04b] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. “Understanding Random SAT: Beyond the Clauses-to-Variables Ratio”. In: *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP’04)*. 2004 (cit. on pp. 50, 53).

[OMa+08] E. O’Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O’Sullivan. “Using Case-Based Reasoning in an Algorithm Portfolio for Constraint Solving”. In: *Proceedings of the Nineteenth Irish Conference on Artificial Intelligence and Cognitive Science (AICS’08)*. 2008 (cit. on pp. 52, 53).

[OHL15] R. J. Oentaryo, S. D. Handoko, and H. C. Lau. “Algorithm Selection via Ranking”. In: *Proceedings of the Twenty-ninth AAAI Conference on Artificial Intelligence (AAAI’15)*. 2015 (cit. on p. 33).

[Özt+22] E. Öztürk, F. Ferreira, H. Jomaa, L. Schmidt-Thieme, J. Grabocka, and F. Hutter. “Zero-Shot AutoML with Pretrained Models”. In: *Proceedings of the 39th International Conference on Machine Learning (ICML’22)*. 2022 (cit. on p. 75).

[Pim+21] N. Pimpalkhare, F. Mora, E. Polgreen, and S. A. Seshia. “MedleySolver: Online SMT algorithm selection”. In: *Proceedings of the Twenty-fourth International Conference on Theory and Applications of Satisfiability Testing (SAT’21)*. 2021 (cit. on p. 130).

[PNK15] S. Pölsterl, N. Navab, and A. Katouzian. “Fast Training of Support Vector Machines for Survival Analysis”. In: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD’15)*. 2015 (cit. on p. 107).

[Pra+21] R. P. Prager, M. V. Seiler, H. Trautmann, and P. Kerschke. “Towards Feature-Free Automated Algorithm Selection for Single-Objective Continuous Black-Box Optimization”. In: *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI’16)*. 2021 (cit. on pp. 54, 56).

[15a] *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*. 2015.

[15b] *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD'15)*. 2015.

[21] *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'21)*. 2021.

[10] *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI'10)*. 2010.

[15c] *Proceedings of the Nineth International Conference on Learning and Intelligent Optimization (LION'15)*. 2015.

[14] *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SOCS'14)*. 2014.

[19] *Proceedings of the Thirty-Third Conference on Artificial Intelligence (AAAI'19)*. 2019.

[Pul+22] D. Pulatov, M. Anastacio, L. Kotthoff, and H. H. Hoos. "Opening the Black Box: Automated Software Analysis for Algorithm Selection". In: *Proceedings of the First International Conference on Automated Machine Learning (AutoML-Conf'22)*. 2022 (cit. on p. 86).

[PK20] D. Pulatov and L. Kotthoff. "Opening the Black Box: Automatically Characterizing Software for Algorithm Selection (Student Abstract)". In: *Proceedings of the Thirty-Fourth Conference on Artificial Intelligence (AAAI'20)*. 2020 (cit. on p. 86).

[PT09] L. Pulina and A. Tacchella. "A Self-Adaptive Multi-Engine Solver for Quantified Boolean Formulas". In: *Constraints* (2009) (cit. on p. 20).

[Ric76] J. R. Rice. "The Algorithm Selection Problem". In: *Advances in Computers* (1976) (cit. on pp. 1, 2, 13, 14).

[RH09] M. Roberts and A. Howe. "Learning from Planner Performance". In: *Artificial Intelligence* (2009) (cit. on p. 57).

[RHF07] M. Roberts, A. Howe, and L. Flom. "Learned Models of Performance for Many Planners". In: *ICAPS: Workshop AI Planning and Learning*. 2007 (cit. on p. 31).

[Rok09] L. Rokach. "Collective-Agreement-Based Pruning of Ensembles". In: *Computational Statistics & Data Analysis* (2009) (cit. on p. 146).

[RdS12] A. L. D. Rossi, A. C. P. L. F. de Carvalho, and C. Soares. "Meta-learning for Periodic Algorithm Selection in Time-Changing Data". In: *Proceedings of the Brazilian Symposium on Neural Networks*. 2012 (cit. on p. 130).

[Ruh+23] T. Ruhkopf, A. Mohan, D. Deng., A. Tornede, F. Hutter, and M. Lindauer. "MASIF: Meta-learned Algorithm Selection using Implicit Fidelity Information". In: *Transactions on Machine Learning Research* (2023) (cit. on p. 174).

[Rus+18] D. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen. "A Tutorial on Thompson Sampling". In: *Foundations and Trends in Machine Learning* (2018) (cit. on p. 116).

[Saa00] D. G. Saari. "The Mathematics of Voting: Democratic Symmetry". In: *Economist* (2000) (cit. on p. 145).

[Sas+22] R. Sass, E. Bergman, A. Biedenkapp, F. Hutter, and M. Lindauer. "DeepCAVE: An Interactive Analysis Tool for Automated Machine Learning". In: *ICML: Workshop on Adaptive Experimental Design and Active Learning in the Real World (ReALML'22)*. 2022 (cit. on p. 24).

[SH18] D. Schäfer and E. Hüllermeier. "Dyad Ranking Using Plackett-Luce Models Based on Joint Feature Representations". In: *Machine Learning* (2018) (cit. on pp. 68, 70).

[Sch90] R. E. Schapire. "The Strength of Weak Learnability". In: *Machine Learning* (1990) (cit. on pp. 142, 147).

[Sch+22] E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney. "A Survey of Methods for Automated Algorithm Configuration". In: *Journal of Artificial Intelligence Research* (2022) (cit. on pp. 2, 22, 62).

[SH79] J. Schmee and J.G. Hahn. "A Simple Method for Regression Analysis with Censored Data". In: *Technometrics* (1979) (cit. on pp. 91, 101–103, 105, 106, 228).

[Sch15] J. Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* (2015) (cit. on p. 54).

[Sch82] P.J. Schoemaker. "The Expected Utility Model: Its Variations, Purposes, Evidence and Limitations". In: *Journal of Economic Literature* (1982) (cit. on p. 96).

[SS01] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001 (cit. on p. 112).

[Scu10] D. Sculley. "Combined Regression and Ranking". In: *The 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'10)*. 2010 (cit. on pp. 34, 172).

[Sei+20] M. Seiler, J. Pohl, J. Bossek, P. Kerschke, and H. Trautmann. "Deep Learning as a Competitive Feature-free Approach for Automated Algorithm Selection on the Traveling Salesperson Problem". In: *Proceedings of the Sixteenth International Conference on Parallel Problem Solving from Nature (PPSN'20)*. 2020 (cit. on pp. 55, 56).

[SL09] J. Sexton and P. Laake. “Standard Errors for Bagged and Random Forest Estimators”. In: *Computational Statistics & Data Analysis* (2009) (cit. on p. 126).

[SH14] A. Shaker and E. Hüllermeier. “Survival Analysis on Data Streams: Analyzing Temporal Events in Dynamically Changing Environments”. In: *International Journal of Applied Mathematics and Computer Science* (2014) (cit. on p. 109).

[SS06] S. Shalev-Shwartz and Y. Singer. “Efficient Learning of Label Ranking by Soft Projections onto Polyhedra”. In: *Journal of Machine Learning Research* (2006) (cit. on p. 34).

[Sie+19] S. Sievers, M. Katz, S. Sohrabi, H. Samulowitz, and P. Ferber. “Deep Learning for Cost-Optimal Planning: Task-Dependent Planner Selection”. In: *Proceedings of the Thirty-Third Conference on Artificial Intelligence (AAAI’19)*. 2019 (cit. on p. 56).

[SB17] D. Sigurdson and V. Bulitko. “Deep Learning for Real-Time Heuristic Search Algorithm Selection”. In: *Proceedings of the Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE’17)*. 2017 (cit. on p. 56).

[Ste+10] D. Stern, H. Samulowitz, R. Herbrich, T. Graepel, L. Pulina, and A. Tacchella. “Collaborative Expert Portfolio Management”. In: *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence (AAAI’10)*. AAAI Press, 2010 (cit. on p. 40).

[SK09] X. Su and T. M. Khoshgoftaar. “A Survey of Collaborative Filtering Techniques”. In: *Advances in Artificial Intelligence* (2009) (cit. on pp. 37–39).

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018 (cit. on p. 126).

[Tho33] W. R. Thompson. “On the Likelihood that one Unknown Probability Exceeds Another in View of the Evidence of Two Samples”. In: *Biometrika* (1933) (cit. on p. 116).

[Tho+13] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms”. In: *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’13)*. 2013 (cit. on pp. 23, 62, 162).

[TM15] K. Tierney and Y. Malitsky. “An Algorithm Selection Benchmark of the Container Pre-Marshalling Problem”. In: *Proceedings of the Nineth International Conference on Learning and Intelligent Optimization (LION’15)*. 2015 (cit. on p. 1).

[TBH22] A. Tornede, V. Bengs, and E. Hüllermeier. “Machine Learning for Online Algorithm Selection under Censored Feedback”. In: *Proceedings of the Thirty-Sixth Conference on Artificial Intelligence (AAAI’22)*. 2022 (cit. on pp. 5, 10, 110).

[Tor+22] A. Tornede, L. Gehring, T. Tornede, M. Wever, and E. Hüllermeier. "Algorithm Selection on a Meta Level". In: *Machine Learning* (2022) (cit. on pp. 5, 10, 137).

[TWH19] A. Tornede, M. Wever, and E. Hüllermeier. "Algorithm Selection as Recommendation: From Collaborative Filtering to Dyad Ranking". In: *29th Workshop Computational Intelligence*. **Young Author Award**, 2019 (cit. on pp. 4, 9, 61).

[TWH20a] A. Tornede, M. Wever, and E. Hüllermeier. "Extreme Algorithm Selection with Dyadic Feature Representation". In: *Proceedings of the International Conference on Discovery Science (DS'20)*. 2020 (cit. on pp. 4, 9, 61).

[TWH20b] A. Tornede, M. Wever, and E. Hüllermeier. "Towards Meta-Algorithm Selection". In: *NeurIPS: Workshop on Meta-Learning (MetaLearn'20)*. 2020 (cit. on pp. 5, 10, 137).

[Tor+20a] A. Tornede, M. Wever, S. Werner, F. Mohr, and E. Hüllermeier. "Run2Survive: A Decision-theoretic Approach to Algorithm Selection based on Survival Analysis". In: *Proceedings of the 12th Asian Conference on Machine Learning (ACML'20)*. 2020 (cit. on pp. 4, 9, 17, 47, 89, 115).

[Tor+23] T. Tornede, A. Tornede, L. Fehring, L. Gehring, H. Graf, J. Hanselle, F. Mohr, and M. Wever. "PyExperimenter: Easily Distribute Experiments and Track Results". In: *Journal of Open Source Software* (2023) (cit. on p. 177).

[Tor+21a] T. Tornede, A. Tornede, J. Hanselle, M. Wever, F. Mohr, and E. Hüllermeier. "Towards Green Automated Machine Learning: Status Quo and Future Directions". In: *arXiv:2111.05850* (2021) (cit. on p. 6).

[Tor+21b] T. Tornede, A. Tornede, M. Wever, and E. Hüllermeier. "Coevolution of Remaining Useful Lifetime Estimation Pipelines for Automated Predictive Maintenance". In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'21)*. 2021 (cit. on p. 24).

[Tor+20b] T. Tornede, A. Tornede, M. Wever, F. Mohr, and E. Hüllermeier. "AutoML for Predictive Maintenance: One Tool to RUL them All". In: *ECML/PKDD: Workshop on IoT Streams for Data-Driven Predictive Maintenance*. 2020 (cit. on p. 24).

[TPV08] G. Tsoumakas, I. Partalas, and I. Vlahavas. "A Taxonomy and Short Review of Ensemble Selection". In: *Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications*. 2008 (cit. on pp. 167, 172).

[van+14] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. "Algorithm Selection on Data Streams". In: *Proceedings of the International Conference on Discovery Science (DS'14)*. 2014 (cit. on p. 130).

[van+15] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren. "Having a Blast: Meta-Learning and Heterogeneous Ensembles for Data Streams". In: *Proceedings of the 15th IEEE International Conference on Data Mining (ICDM'15)*. 2015 (cit. on p. 130).

[vDB18] S. van Rijn, C. Doerr, and T. Bäck. “Towards an Adaptive CMA-ES Configurator”. In: *Proceedings of the 15th International Conference on Parallel Problem Solving from Nature (PPSN’18)*. 2018 (cit. on p. 130).

[Van18] J. Vanschoren. “Meta-Learning: A Survey”. In: *arXiv:1810.03548* (2018) (cit. on pp. 22, 53, 139).

[Van+13] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. “OpenML: Networked Science in Machine Learning”. In: *SIGKDD Explorations* (2013) (cit. on pp. 72, 73).

[Van10] Joaquin Vanschoren. “Understanding Machine Learning Performance with Experiment Databases”. PhD thesis. 2010 (cit. on p. 86).

[Vas+20] S. Vaswani, A. Mehrabian, A. Durand, and B. Kveton. “Old Dog Learns New Tricks: Randomized UCB for Bandit Problems”. In: *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS’20)*. 2020 (cit. on p. 115).

[VG10] S. Vembu and T. Gärtner. “Label Ranking Algorithms: A Survey”. In: *Preference Learning*. Springer, 2010 (cit. on p. 32).

[VR17] G. Verbruggen and L. De Raedt. “Towards automated relational data wrangling”. In: *ECML-PKDD: Workshop on Automatic Selection, Configuration and Composition of Machine Learning Algorithms* (2017) (cit. on p. 23).

[VGB09] R. Vilalta, C. Giraud-Carrier, and P. Brazdil. “Meta-Learning-Concepts and Techniques”. In: *Data Mining and Knowledge Discovery Handbook*. Springer, 2009 (cit. on p. 139).

[Wag+18] M. Wagner, M. Lindauer, M. Misir, S. Nallaperuma, and F. Hutter. “A Case Study of Algorithm Selection for the Traveling Thief Problem”. In: *Journal of Heuristics* (2018) (cit. on p. 163).

[Wan+19] W. Wang, V. W. Zheng, H. Yu, and C. Miao. “A Survey of Zero-Shot Learning: Settings, Methods, and Applications”. In: *ACM Transactions on Intelligent Systems and Technology* (2019) (cit. on p. 63).

[Wan+13] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, and T. Liu. “A Theoretical Analysis of NDCG Ranking Measures”. In: *Proceedings of the 26th Annual Conference on Learning Theory (COLT’13)*. 2013 (cit. on pp. 41, 76).

[Wan+20] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. “Generalizing From a Few Examples: A Survey on Few-Shot Learning”. In: *ACM Computing Surveys* (2020) (cit. on p. 22).

[Wei+07] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. “COFI RANK - Maximum Margin Matrix Factorization for Collaborative Ranking”. In: *Proceedings of the 20th International Conference on Advances in Neural Information Processing Systems (NeurIPS’07)*. 2007 (cit. on pp. 40, 76).

[WKW16] K. Weiss, T. M. Khoshgoftaar, and D. Wang. “A Survey of Transfer Learning”. In: *Journal of Big Data* (2016) (cit. on pp. 22, 173).

[Wes16] S. Wessing. “Towards a Systematic Development Process of Optimization Methods”. In: *arXiv:1603.00001* (2016) (cit. on p. 174).

[Wev+19] M. Wever, F. Mohr, A. Tornede, and E. Hüllermeier. “Automating Multi-Label Classification Extending ML-Plan”. In: *ICML: Workshop on Automated Machine Learning*. 2019 (cit. on p. 24).

[Wev+21] M. Wever, A. Tornede, F. Mohr, and E. Hüllermeier. “AutoML for Multi-Label Classification: Overview and Empirical Evaluation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021) (cit. on p. 24).

[Whi+21] C. White, A. Zela, R. Ru, Y. Liu, and F. Hutter. “How Powerful Are Performance Predictors in Neural Architecture Search?” In: *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS’21)*. 2021 (cit. on p. 86).

[Wol92] D. H. Wolpert. “Stacked Generalization”. In: *Neural Networks* (1992) (cit. on pp. 40, 142).

[WM97] D. H. Wolpert and W. G. Macready. “No Free Lunch Theorems for Optimization”. In: *Evolutionary Computation* (1997) (cit. on p. 1).

[Xu+11] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Hydra-MIP: Automated Algorithm Configuration and Selection for Mixed Integer Programming”. In: *IJCAI: International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*. 2011 (cit. on pp. 29, 102).

[Xu+08] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. “SATzilla: Portfolio-Based Algorithm Selection for SAT”. In: *Journal of Artificial Intelligence Research* (2008) (cit. on pp. 31, 53, 91, 105, 112).

[Yan+19] C. Yang, Y. Akimoto, D. W. Kim, and M. Udell. “OBOE: Collaborative Filtering for AutoML Model Selection”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD’19)*. 2019 (cit. on p. 86).

[Zha+21] K. Zhao, S. Liu, J. X. Yu, and Y. Rong. “Towards Feature-free TSP Solver Selection: A Deep Learning Approach”. In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN’21)*. 2021 (cit. on p. 56).

[Zöl+22] M. Zöller, W. Titov, Thomas T. Schlegel, and M. F. Huber. “XAutoML: A Visual Analytics Tool for Establishing Trust in Automated Machine Learning”. In: *arXiv:2202.11954* (2022) (cit. on p. 24).

List of Symbols

$\mathcal{A}$	Set of candidate algorithms
a	Candidate algorithm
$\mathcal{I}$	Problem instance space
$\mathcal{I}_D$	Set of training instances
i	Problem instance
$\boldsymbol{f}_i$	Problem instance features
f	Problem instance feature function
$\boldsymbol{f}_a^A$	Algorithm features
f^A	Algorithm feature function
l	Loss function
$\widehat{l}$	Loss function surrogate
s	Algorithm selector
$\mathcal{S}$	Set of algorithm selectors
ass	Algorithm selector selector
agg	Aggregation function
s_{online}	Online algorithm selector
s^*	Oracle / VBS
s^*_{online}	Online oracle
$\mathcal{H}$	Selection history space
h	Selection history
C	Cutoff / Timeout time
Run2Survive	Approach to solve the AS problem based on survival analysis.
Run2SurviveExp	Variant of Run2Survive.
Run2SurvivePAR10	Variant of Run2Survive.
Run2SurvivePoly/Log	Variant of Run2Survive.
BlindUCB	UCB variant ignoring censored data.
BClinUCB	UCB variant with a bias correction.
$\mathscr{L}$	Loss function mapping runtimes to real-valued loss degrees

List of Abbreviations

AS algorithm selection

OAS online algorithm selection

VBS virtual best solver

SBS single best solver

MetaAS meta algorithm selection

CNN convolutional neural network

RNN recurrent neural network

ELA exploratory landscape analysis

XAS extreme algorithm selection

CASH combined algorithm selection and hyperparameter optimization

AutoML automated machine learning

ML machine learning

NAS neural architecture search

AC algorithm configuration

HPO hyperparameter optimization

CF collaborative filtering

PL Plackett-Luce

SA survival analysis

ASlib algorithm selection library

MAB multi-armed bandits

CPMP container premarshalling problem

VRP vehicle routing problem

DAC dynamic algorithm configuration

RR ridge regression

SAT boolean satisfiability

TSP traveling salesperson

CSP constraint satisfaction problem

ASP answer set programming

BNSL bayesian network structured learning

SGI subgraph isomorphism

MAXSAT maximum satisfibility problem

MIP mixed integer programming

QBF quantified boolean formula

TTP traveling thief problem

CHF cumulative hazard function

List of Figures

List of Tables

Colophon

This thesis was typeset with LaTeX 2$_\varepsilon$. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.