Dissertation zur Erlangung des Doktorgrades der Technischen Fakultät der Albert-Ludwigs-Universität Freiburg im Breisgau

Dynamic Algorithm Configuration by Reinforcement Learning

André Biedenkapp

2022

Dean:
Prof. Dr. Roland Zengerle, *University of Freiburg, Germany*

PhD advisor and first reviewer:
Prof. Dr. Frank Hutter, *University of Freiburg, Germany*

Second PhD advisor:
Prof. Dr. Marius Lindauer, *Leibniz University Hannover, Germany*

Second reviewer:
Dr.-ing. habil. Carola Doerr, *Sorbonne Université, CNRS, LIP6, France*

Date of defense:
14.10.2022

Dynamic Algorithm Configuration

Abstract

The performance of algorithms, be it in the domain of machine learning, hard combinatorial problem solving or AI in general depends on their many parameters. Tuning an algorithm manually, however, is error-prone and very time-consuming. Many, if not most, algorithms are iterative in nature. Thus, they traverse a potentially diverse solution space, which might require different parameter settings at different stages to behave optimally. Further, algorithms are often used for solving a diverse set of problem instances, which by themselves might require different parameters. Taking all of this into account is infeasible for a human designer. Automated methods have therefore been proposed to mitigate human errors and minimize manual efforts. While such meta-algorithmic methods have shown large successes, there is still a lot of untapped potentials as prior approaches typically only consider configurations that do not change during an algorithm's run or do not adapt to the problem instance.

In this dissertation, we present the first framework that is capable of dynamically configuring algorithms, in other words, capable of adapting configurations to the problem instance at hand during an algorithm's solving process. To this end, we model the *dynamic algorithm configuration (DAC)* problem as a contextual Markov decision process. This enables us to learn dynamic configuration policies in a data-driven way by means of reinforcement learning.

We empirically demonstrate the effectiveness of our framework on a diverse set of problem settings consisting of artificial benchmarks, evolutionary algorithms, AI planning systems, as well as deep learning. We show that DAC outperforms previous meta-algorithmic approaches. Building on these successes, we formulate the first standardized interface for dynamic configuration and an extensive benchmark to facilitate reproducibility and lower the barrier of entry for new researchers into this novel research field. Lastly, our work on DAC feeds back into the reinforcement learning paradigm. Through the lens of DAC, we identify shortcomings in current state-of-the-art approaches and demonstrate how to solve these. In particular, intending to learn general policies for DAC, our work pushes the boundaries of generalization in reinforcement learning. We demonstrate how to efficiently incorporate domain knowledge when training general agents and propose to move from a reactive way of doing reinforcement learning to a proactive way by learning when to make new decisions.

Zusammenfassung

Die Leistungsfähigkeit von Algorithmen im Bereich des maschinellen Lernens, kombinatorischer Optimierungsprobleme oder der künstlichen Intelligenz im Allgemeinen, hängt von ihren vielen Parametern ab. Das manuelle Konfigurieren eines Algorithmus ist jedoch fehleranfällig und sehr zeitaufwendig. Des weiteren sind viele, wenn nicht sogar die meisten, Algorithmen iterativer Natur. Daher durchlaufen sie einen potenziell vielfältigen Lösungsraum, der in verschiedenen Phasen unterschiedliche Parametereinstellungen erfordern kann, um optimal gelöst zu werden. Darüber hinaus werden Algorithmen häufig zur Lösung einer Vielzahl von Problemfällen eingesetzt, die ihrerseits unterschiedliche Parameter erfordern können. All dies zu berücksichtigen, ist für einen menschlichen Designer nicht machbar. Um menschliche Fehler zu vermeiden und den manuellen Aufwand zu minimieren, wurden automatisierte Methoden entwickelt, die diese Aufgabe übernehmen. Während solche meta-algorithmischen Methoden enorme Erfolge gezeigt haben, gibt es jedoch noch viel ungenutztes Potenzial, da bisherige Ansätze typischerweise nur Konfigurationen berücksichtigen, die sich während der Laufzeit eines Algorithmus nicht ändern oder sich nicht an die Probleminstanz anpassen.

In dieser Dissertation stellen wir das erste Framework vor, das in der Lage ist, Algorithmen dynamisch zu konfigurieren, d. h. die Konfigurationen während des Lösungsprozesses eines Algorithmus an die jeweilige Probleminstanz anzupassen. Zu diesem Zweck modellieren wir das Problem der dynamischen Algorithmenkonfiguration (*dynamic algorithm configuration* – DAC) als einen kontexterweiterten Markov-Entscheidungsprozess. Dies ermöglicht es uns, dynamische Konfigurationen mittels Reinforcement Learning zu erlernen.

Wir demonstrieren empirisch die Effektivität unseres Frameworks auf einer Vielzahl von Problemstellungen, bestehend aus künstlichen White-Box Benchmarks, evolutionären Algorithmen, Planungssystemen sowie Deep Learning und zeigen, dass DAC bisherige meta-algorithmische Ansätze übertrifft. Aufbauend auf diesen Erfolgen formulieren wir das erste standardisierte Interface für dynamische Konfiguration und präsentieren eine umfangreiche Benchmark, die Reproduzierbarkeit erleichtert und die Einstiegshürde für neue Forscher in dieses neue Forschungsfeld senkt. Schließlich fließt unsere Arbeit an DAC in das Paradigma des Reinforcement Learning zurück. Aus der Sichtweise von DAC identifizieren wir Mängel in den aktuellen State-of-the-Art-Ansätzen und zeigen, wie diese gelöst werden können. Insbesondere mit der Absicht allgemeine Strategien für DAC zu lernen, verschiebt unsere Arbeit die Grenzen der Generalität von Reinforcement Learning Methoden. Wir zeigen, wie man beim Training allgemeiner Agenten effizient Domänenwissen einbeziehen kann. Weiter schlagen wir vor, von einer reaktiven Art des Reinforcement Learning zu einer proaktiven Art überzugehen, in der gelernt wird wann neue Entscheidungen getroffen werden müssen.

Acknowledgments

I consider myself unbelievably lucky for the opportunity to have worked with so many incredible people without whom this dissertation would not have been possible. First and foremost I want to thank my two advisors Frank Hutter and Marius Lindauer. You created an incredibly enriching environment in which I felt comfortable and wholeheartedly enjoyed my time as a Ph.D. student. It is phenomenally empowering that you always take feedback to heart which allows every member to help shape the environment to our collective needs. In times of setbacks, you kept me motivated and taught me that even these moments provide new opportunities. You helped me become the researcher that I am today. For that, I will always be grateful.

I also want to thank Katharina Eggensperger, Matthias Feurer, Aaron Klein and Stefan Falkner. You welcomed me with open arms to the group and treated me as a colleague from day one. It was a pleasure to start my Ph.D. journey with you. I am especially grateful to Katharina. You are the best office mate one could hope for. From "babysitting" me at my first conference to always having my back in times when I doubted myself. You are a great researcher, mentor, and friend.

Still, there are many more members that joined the Freiburg group later and whom I am thankful to have had the opportunity to work with: Arbër, Raghu, Robin, Thomas, Steven, Noor, Mahmoud, Fabio, Jörg, Neeratyoy, Samuel, Frederic, Danny, Rhea, and Eddie. Though a pandemic forced us to work remotely, it was a pleasure to work alongside you.

I further want to extend my thanks to all the fantastic students I got to work with. Baohe Zhang, Gresa Shala, H. Furkan Bozkurt, Oliver Brunner and Andreas "Andy" Sälinger. It was a pleasure to work with you on your projects and to see your creativity in action. Special thanks go out to Andy for all the fun and engaging discussions about RL.

I would be amiss to not mention my fantastic collaborators from Hannover: Theresa Eimer, Carolin Benjamins, Difan Deng, René Sass, and Aditya Mohan. Thank you all for sharing your passion for contextual reinforcement learning and AutoML with me. I particularly would like to thank Theresa who has co-authored many papers with me already. Through it all, I learned a lot from you and am glad to be able to not only call you a colleague but a friend.

I would like to thank each and every one of my co-authors: Steven Adriaensen, Noor H. Awad, Carolin Benjamins, Eddie Bergman, H. Furkan Bozkurt, Roberto Calandra, Kurtland Chua, Nguyen Dang, Difan Deng, Carola Doerr, Katharina Eggensperger, Theresa Eimer, Thomas Elsken, Aleksandra Faust, Chris Fawcett, Matthias Feurer, Jörg K. H. Franke, Julia Guerrero-Viu, Sven Hauns, Holger H. Hoos, Frank Hutter, Sergio Izquierdo, Gregor Köhler, Martin S. Krejca, Nathan O. Lambert, Marius Lindauer, Neeratyoy Mallik, Joshua Marben, Robert Mattmüller, Yingjie Miao, Guilherme Miotto, Aditya Mohan, Philipp Müller, Samuel Müller, Vu Nguyen, Jack Parker-Holder, Luis Pineda, Raghu Rajan, Maximilian Reimer, Bodo Rosenhahn, Tim Ruhkopf, René Sass, Simon Schrodi, Frederik Schubert, Jendrik

Seipp, Gresa Shala, Silvan Sievers, Xingyou Song, David Speck, Diederick Vermetten, Hao Wang and Baohe Zhang. Thank you all for sharing your expertise with me and furthering my passion for research.

Besides fantastic colleagues, I am incredibly lucky to be able to count on a strong support system. I could always count on my friends Rick Gelhausen, David Speck, Lukas Gemein, and Julian Kunzelmann. We already shared the joys and pains of bachelor's and/or master's degrees. I am glad to have also shared my Ph.D. journey with you. I am thankful for all the strolls through the city or the park and all the zoom calls we shared over the years. I was incredibly happy to have had you all by my side on my wedding day. I am looking forward to sharing and creating many more memories with you all.

My family was always there for me and always helped me get back up even after the biggest setbacks. My parents inspired and encouraged me from a young age to ask questions and explore the world. Without their encouragement and that of my big sister, I might have faltered and not even dared to dream of a Ph.D. Thank you for encouraging me to dream and to help make that dream a reality. I also want to thank my mother-in-law who encouraged and supported me from the moment we met.

Last but not least, I am forever grateful to my incredibly loving and beautiful wife, Kristin Biedenkapp. I am forever happy to have met you. Your kindness, love, and support are what kept and keep me going. If I could find the right words I would write pages upon pages to express my gratitude and love for you. Alas, I'm no poet. All I can say is, I love you.

Contents

Part I

Introduction

CHAPTER 1

Motivation

The vast field of *artificial intelligence (AI)* has made impressive progress in a variety of fields, such as satisfiability solving (SAT; see, e. g., Biere et al., 2009), AI planning (see, e. g., Ghallab et al., 2004), answer set programming (ASP; see, e. g., Dimopoulos et al., 1997), machine learning (ML; see, e. g., Solomonoff, 1957), reinforcement learning (RL; see, e. g., Sutton and Barto, 2018) and, most recently and prominently, in deep learning (DL; see, e. g., Goodfellow et al., 2016). At the heart of this progress lie the tireless efforts of many researchers aiming to push the boundaries of what is possible with compute power. On the one hand, researchers are designing ever more powerful computer hardware. On the other hand, researchers optimize and develop new algorithms that can efficiently[1] solve ever more challenging problems.

Take for example chess engines. While in the 1950s the first chess computers were easily beaten even by beginner players (Heath and Allum, 1997), in 1997 the expert system Deep Blue (Campbell et al., 2002) was capable of beating the reining world champion using a purpose-built super-computer. Nowadays, chess engines are just as capable but use conventional personal computers. This impressive feat can be both attributed to better software as well as hardware.

In a similar fashion to the chess example above, challenging benchmarks have been used for decades to advance research progress, which has resulted in a plethora of algorithms from which one can choose when aiming to solve a novel problem. However, this is not as straightforward a task as it first might seem. Experts spent months, years, and sometimes even decades developing algorithms that are tailored to solve particular problems. While experts develop understanding and intuition on what works for which type of problem, the typical users often lack this expertise. The question of which algorithm to use for which problem instance is further complicated by the configurability of algorithms. The parameters of algorithms determine how exactly the algorithm will be executed. Correctly configuring the algorithm is crucial to unlocking peak performances. However, this is not trivially done. Thus, users often make uninformed decisions and take the latest state-of-the-art (SOTA) algorithm with some default configuration, without paying attention to how and on which problems the "SOTAness" of the algorithm was determined.

Considerable research efforts have been made to automate and simplify such choices to reduce human effort and mitigate errors. In this line of research, this dissertation proposes a novel framework that can select and adapt algorithms, not only to the problem at hand but also on-the-fly. In the following, we will motivate this line of research and highlight how the novel framework improves over previous frameworks using an illustrative example.

[1] For some notion of efficient.

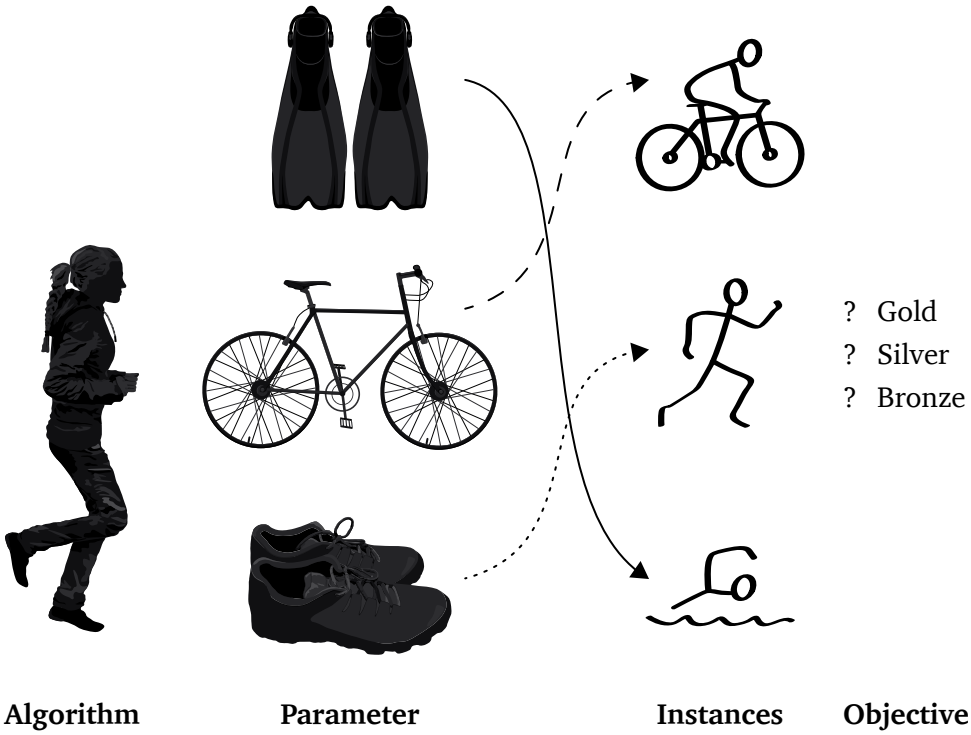


Figure 1.1: An athlete needs to select from different sporting equipment to achieve as many gold medals as possible.

Which Algorithm to Use? Consider the case of an athlete that wants to compete in various sporting disciplines. Their goal is to win as many gold medals as possible. Let us assume the different disciplines are running, swimming and biking. The athlete has different sporting equipment at their disposal, such as a bike, swimming flippers, and running shoes. All of the different sporting equipment will lead to gold medals if they are used in the correct sporting discipline. A visual depiction of this problem setup is given in Figure 1.1. However, wrong choices will often not even result in a bronze medal. For example, trying to run as fast as possible using swimming flippers will be much slower than using running shoes. While this example might seem ridiculous to some readers, we believe this exaggerated example is not too far from how algorithms are often applied in practice. If a user has no prior knowledge that they can use to make an informed decision about what is adequate or good sporting equipment, they might end up trying to run using swimming flippers.

To mitigate this problem of manually selecting the best algorithm[2], already in 1976, Rice proposed to automate this problem through the use of *algorithm selection* (AS; Rice, 1976). Instead of relying on human labor to build the expertise by trial-and-error, AS makes use of machine learning to train a so-called *selector* to decide which algorithm to use for a particular problem instance. The selector is trained to learn a mapping from problem instance to algorithm similar to how a human practitioner might make their decision. Based on previously recorded performance data of the potential algorithms and some characteristic features of the problem instance at hand, the selector learns which mapping will lead to the best result on the training data. The resulting selector is then able to make informed decisions about which algorithm to use, given that future instances resemble those the selector was trained on.

AS already enables the athlete to solve the previous example. After the athlete has

[2]According to some performance metric

tried out the different equipment for each task they have perfect knowledge about the usefulness of the equipment. For example, using the bike in the running task will lead to disqualification and the flippers to a non-competitive time. Using their expert knowledge they can win many gold medals in various disciplines, as long as the disciplines are similar to those their expert knowledge was built on. Otherwise, the selector might not be able to correctly assign the correct sporting equipment to the new task.

What About Algorithm Parameters? The example so far oversimplified the problem. The athlete was faced with very distinct choices of which sporting equipment to use for distinct sporting disciplines. In practice, the choices are often far less straightforward. For example, novel search techniques or heuristics can be incorporated into existing algorithms, e. g., for SAT solving or AI planning. The choice of which to use is often left to the user through the use of parameters. Far worse even, parameter values are often not a finite set of discrete choices but are numerical and span an infinite range of possibilites.

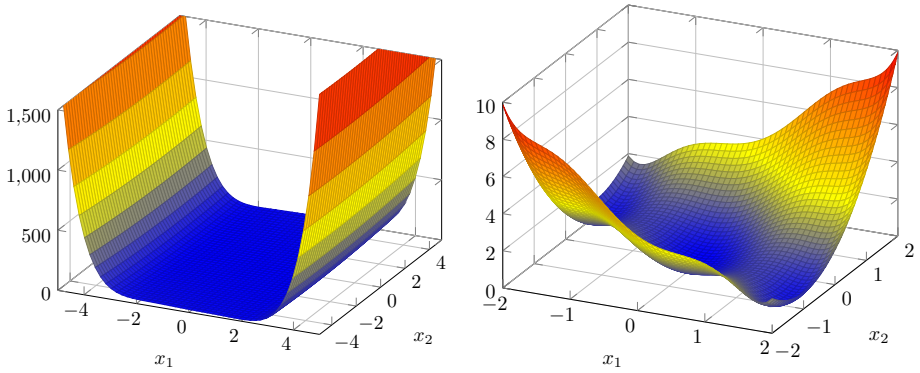
To reflect this, we can adapt our example and, for now, focus only on the biking task. The athlete is still faced with the choice of which sporting equipment to use. Now, however, the equipment itself is variable. In other words, besides a categorical choice parameter to determine the equipment, there are additional parameters to configure the equipment itself. For example, consider that the bike's tires have to be pressurized by the athlete. If the pressure is too low the athlete risks that the tire gets punctured, stopping them from competing in the race. On the other hand, if the pressure is too high the bike ride will be highly uncomfortable and slower as the overinflated tire will have much lower traction (see Figure 1.2). Using AS the athlete would need to first try out all infinitely many possible choices of pressure before they could make an informed choice. This, however, is most often infeasible in practice[3].

Algorithm configuration (AC; Hutter et al., 2009) alleviates users from having to consider potentially infinitely many parameter values. To do so, a *configurator* tries out different parameter values in the potentially infinite search space. Similar to AS, this generated experience about the algorithm's performance can be used to select which configuration, i.e. parameter values, to choose. Counter to AS methods, AC methods iterate and extend this knowledge to better probe the space in an informed way. This is typically done by building predictive models that can be used to estimate which configurations will lead to good performance without having to test every possible parameter value. The most promising configurations (according to the predictive model) will be tested to further extend the knowledge and refine the model. A crucial difference between AC and AS is that AC recommends a single configuration for a set of similar tasks and does not learn a mapping from instance to configuration. Thus, AC would not be able to solve the full example with highly different sporting disciplines.

Figure 1.2: Different tire pressures. Top: Under pressurized/flat tire. Middle: Tire with appropriate pressure and good traction. Bottom: Over pressurized tire with low traction.

Making use of AC, the athlete now only needs to try out a few tire pressures before they can make an informed decision about which to use for the biking task. To solve the full example, the athlete can combine AC with AS to figure out what are the best parameters for the chosen

[3]Note that there are meaningful discretizations for this example to make AS feasible. For algorithm parameters, this is often not trivially done.



(a) Global View (b) Local View

Figure 1.3: Different views of the *Three-Hump Camel* function. Note that (b) is only a zoomed-in version of (a). The orientation of the plots is the same.

sporting equipment even for dissimilar tasks. This combination is referred to as *per-instance algorithm configuration* (PIAC; L. Xu et al., 2010). In essence, PIAC uses AC to find a finite set of well-performing, complementary configurations which ideally work well for different problem instances. PIAC then uses AS to select which of the possible configurations to use for a task at hand. Thus, the athlete can now set up their sporting equipment perfect for the task at hand even when faced with infinitely many possible choices, for a variety of problem instances.

Algorithms in the Real World Again, the example so far is oversimplifying the problem. The athlete is still faced with distinct tasks where the goal and how to reach it are abundantly clear. When aiming for gold in each task, the athlete has to only perform a single activity as fast as possible. However, in practice, algorithms are often not faced with such straightforward tasks. This does not mean that AS, AC, and PIAC are not useful approaches to optimizing algorithms. They have shown tremendous successes, as we will discuss in Chapter 3. However, there is room for further improvement by taking the algorithm's behavior into account.

Many if not most algorithms are iterative in nature. By design, this causes algorithms to move through a solution space. In practice, the global solution space might look very different from local patches of the space (see, e. g., Figure 1.3) and thus potentially require different parameter values. We can again reflect this in the running example, where the athlete now also wants to compete in biathlons or triathlons. If the athlete were to only use a single sporting equipment for a whole triathlon they surely would not win the competition. Based on the stage of the triathlon the athlete will have to switch from running shoes to a bike to swimming gear. Note that it is not enough to know which equipment to use but that it also matters when to use which equipment.

The previously discussed methods for optimizing algorithms do help with this problem to some degree. If only a fixed choice is used over the whole run of the algorithm by AS, AC or PIAC then the choice will be the best on average. For example, while taking the bike into the water or trying to run with flippers will result in bad performance for the athlete, they can at least successfully participate in the triathlon only by using the running shoe. It will not be optimal and not lead to a win but is the single best static option in the example. The methods could also be used to optimize a *parameterized schedule*. For example, the order of equipment usage could be optimized, or even the length of equipment use. Still, this has potential failure modes. Imagine that a last-minute change to the triathlon results in a drastically different course and order of tasks, then sticking to the predetermined

schedule will again result in bad performance of the athlete. If, however, the athlete can make use of their prior knowledge and current observations to decide *on-the-fly* and *for the problem at hand* which equipment to use they will be capable of competing and winning in different conditions and competitions.

Beyond Black Boxes To achieve on-the-fly configuration for the problem at hand we have to reevaluate how the previous approaches treated the problem of configuration. The previously discussed frameworks treated the algorithm as a black box and were content with only considering the input-output relationship of configuration to final performance. Instead, to open the black box, we propose to take the algorithm's step-wise behavior into account, as well as information about the problem at hand to make more informed decisions on-the-fly. Our novel meta-algorithmic framework which we dub *dynamic algorithm configuration* (DAC)[4] is a generalization of the previously mentioned methods and opens up a new research field on meta-algorithmics research. To this end, at each step, DAC observes how an algorithm interacts with the problem space. This allows DAC to observe and identify how quickly an algorithm is progressing or if it is stalling. These observations can be used to decide if an algorithm's configuration has to be changed to induce a different behavior which ideally will result in better progress. In the running example, this translates to the athlete observing their surroundings and based on changes deciding which equipment to use. For example, when the athlete is about to enter the water for the swimming portion of the triathlon this should trigger the change in equipment to the swimming gear.

1.1 In a Nutshell

In this dissertation, we focus on the problem of how to configure algorithms on-the-fly and to the problem at hand, to make them as performant as possible. This work enables improvements of algorithms in a broad variety of fields, leading to potentially much more efficient algorithms. In particular, we will present case studies in the domains of evolutionary algorithms and AI planning, and also give an example for deep learning. We show that DAC is not only an improvement in theory but also provide ample empirical evidence for its success. Finally, we discuss why modern reinforcement learning (RL) methods are suitable solution approaches to DAC and how research on DAC enables further advancements in RL, in particular for generalization through the use of context information.

1.2 How to Read This Dissertation

This dissertation is a *cumulative dissertation (dissertation by publication)*, as such, it consolidates previously published works by the author all of which aim to answer a common scientific question. For improved readability, we structured this dissertation into six parts which are sorted by topic rather than chronologically. Part I (Introduction) provides the high-level motivation and concrete goals for our research. It further provides the relevant background and extends the discussion on related work, of our previous publications, with a focus on automated machine learning. Part II (Dynamic Algorithm Configuration: The Problem), Part III (Dynamic Algorithm Configuration: Case Studies), Part IV (Dynamic Algorithm Configuration: Benchmarking) and Part V (Improving RL From the Lens of DAC) contain two to three chapters, each of which consist of one publication. Part VI (Conclusion) then closes the dissertation with an extensive summary and discussion of our contributions and provides an outlook on potential future work. Each chapter is self-contained, provides

[4] Pronounced like the English word duck (/dʌk/).

overviews of relevant literature, and could be read independently. Still, as this dissertation introduces a novel research field for meta-algorithmics we suggest readers first read Parts I and II before moving to other parts.

CHAPTER 2

Goals of this Dissertation

The continuous growth in AI and ML research results in a seemingly unending growth in development of algorithms and algorithm components with vast application areas. Still, the expert knowledge required to make efficient use of these methods is often only held by a small number of researchers in the particular field while the methods could be beneficial for many non-experts. This often hinders the cross-pollination of ideas between different research fields. Lowering the barrier of entry by providing methods to simplify the use of algorithms while being as efficient and high-performing as possible is crucial to further advancing AI research in many fields. To make strides towards truly democratized machine learning and AI we believe that learned configuration policies are a crucial component. Thus, with this dissertation we set out to answer the question: *Can we learn general dynamic configuration policies using reinforcement learning?* Towards this goal, we identified the following key challenges that we aim to address with this dissertation.

2.1 Key Challenges

Challenge 1: Formal Problem Definition

Many communities have considered dynamic configuration to improve the performance of algorithms in their domain. One of the best-known examples stems from the deep learning literature. Learning rates need to be changed dynamically as static learning rates can lead to sub-optimal training results or, in the worst case, diverging behavior (Bach and Moulines, 2011). Efforts for dynamic configuration thus far, however, focused mostly on manually designed heuristics. The few approaches that are aimed at automating dynamic configuration typically are application specific and do not consider the general dynamic configuration problem. To consolidate these lines of research and to ensure a joint research framework the first challenge consists of providing a formal problem definition. This enables us to study how the dynamic configuration problem relates to optimization problems. Further, a joint framework makes this line of research more easily accessible and allows to better identify and discuss open problems that individual solution approaches might exhibit.

Challenge 2: Application in the Real World

For an emerging research field, theoretical considerations alone might not be enough to fully understand the problem as the theoretical setting might abstract away issues that can arise in practice. Thus, studying solution approaches on real-world benchmarks is crucial in deepening the understanding of the problem as well as different solution approaches.

By its nature, this challenge blends engineering questions with research questions. For example, questions, such as "How can algorithm X be dynamically configured?" and "How should algorithm X be interfaced with the dynamic configurator?" go hand in hand. Similarly, the question of how to incorporate expert knowledge into a learned dynamic configuration procedure is both a scientific and an engineering question.

Challenge 3: Reproducible Research

To facilitate meaningful empirical comparison between different solution approaches, it is crucial to provide standardized implementations of baselines and benchmarks. Previous research efforts often have designed problem-specific interfaces which do not facilitate easy comparison between novel methods. Further, code is often not made publicly available or only partially available. This creates a huge burden on researchers wanting to compare their methods to existing ones. With the introduction of a new research field, we have the opportunity to build a community with a focus on reproducible research. Thus from the outset, we want to tackle the issues listed above. Altogether the challenge is threefold:

- A standard interface for benchmarks is needed to lower the barrier of entry for researchers.
- To meaningfully compare solution approaches, a diverse set of benchmarks is needed. These should cover cheap-to-run toy benchmarks up to more complex problem settings.
- Whenever possible, found solutions should be open-sourced and made publicly available to lower the cost of comparison for novel research.

Challenge 4: Generalization in Reinforcement Learning

Based on the problem definition, we identify reinforcement learning (RL) as a highly suitable solution approach to dynamic algorithm configuration. The typical RL setting is however not concerned with learning agents that can tackle a broader variety of problem settings. The most commonly studied problem in RL is arguably game-playing (see, e. g., Mnih et al., 2015). In this setting, an RL agent typically has to learn to solve a single level of a game or individual games. In the DAC setting, however, we are intending to learn policies that can configure algorithms on-the-fly as well as per-problem instance. Thus, instead of learning only to play a single game or level, we are interested in learning policies that are capable of solving multiple such problems. When aiming to use modern deep RL methods for dynamic algorithm configuration, we have to tackle the problem of generalization in RL.

2.2 Contributions

In this section, we discuss the contributions of this dissertation. We first use the key challenges laid out before to define research questions we aim to answer. We then provide a high-level summary of how the parts of the dissertation address these questions. Finally, as this work is a cumulative dissertation, we provide individual summaries for each chapter that contains previously published works.

Research Questions

Based on the challenges above, we pose the following research questions:

- What is the current state of dynamic configuration research?

- What is the formal problem that is being solved?
- How can we make use of reinforcement learning to dynamically configure algorithms?
- How can we make use of prior knowledge when learning novel configuration policies?
- How can we make this line of research more reproducible?
- How can we facilitate generalization in reinforcement learning?
- How does the dynamic configuration problem differ from standard reinforcement learning problems?

To answer the first question, each chapter of Parts II to V provides a discussion on related research. Out of these Chapter 6 provides the most extensive and up-to-date discussion and puts our research in the historic context of related dynamic configuration research as well as provides an overview of the related meta-algorithmic problems. Building on this knowledge, the second research question of a formal problem definition is discussed in Part II. The chapters of this part further provide the first examples of how to use modern deep reinforcement learning method to solve the novel dynamic algorithm configuration problem. In particular, the idea of contextual reinforcement learning is explored as a solution approach to the DAC problem throughout this dissertation. Part III provides in-depth representative case studies to give detailed answers on how to use reinforcement learning for dynamic configuration of an AI planning as well as evolutionary algorithms. The former considers configuration in a continuous configuration space and further explores how to incorporate prior knowledge into learned policies, whereas the latter considers a discrete configuration space. By the nature of the DAC problem and the chosen solution approaches, both chapters also explore generalization in RL. Based on the insights from the first empirical applications, Part IV considers how to make this line of research more reproducible by providing a collection of dynamic configuration benchmarks.

Having provided a thorough examination of the DAC problem and example applications, in Part V we set the focus on reinforcement learning and how to improve it from the lens of DAC. We first discuss how reinforcement learning can be made more proactive, which, in essence, uses ideas from DAC to configure action repetitions. We further discuss a curriculum learning approach for faster and better generalization capabilities of reinforcement learning agents. Lastly, we propose a novel benchmark for contextual reinforcement learning to further explore and analyze generalization in RL without additional confounding factors introduced by the DAC problem.

By nature of being a cumulative dissertation, each chapter of the above-summarized parts consists of a previously published work. In the following, we provide individual summaries of these chapters which showcase how these works address the common scientific question of the dissertation and the above-stated research questions.

Chapter 5: Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework

This chapter provides the cornerstone of the dissertation. It provides the first formal problem definition of dynamic algorithm configuration (DAC). This definition considers both on-the-fly as well as per-instance configuration. This is in contrast to previous research efforts which typically either considered the problem only from the per-instance(-set) angle or only from the on-the-fly angle. Combining both angles results in a more flexible problem definition that generalizes over the previous approaches. Informally, we discuss how the DAC problem relates to previous meta-algorithmic frameworks, such as algorithm selection, algorithm configuration, and per-instance algorithm configuration.

In this chapter, we further propose to model the dynamic configuration problem as contextual Markov decision process (cMDP; Hallak et al., 2015). We discuss each part of the cMDP in detail and discuss how the notion of context influences the learning paradigm. We then propose to include the context information of the cMDP as part of the state, such that an RL agent can directly access it and make use of it to learn general policies. This cMDP formulation enables us to make use of modern deep reinforcement learning methods as solution methods for DAC problems. In turn, this makes DAC an interesting problem to study generalization of RL approaches as, in DAC, the goal is to learn configuration policies across multiple problem instances.

To empirically study the DAC problem and RL as a solution approach, we introduce cheap-to-run, controllable artificial benchmarks. These benchmarks allow us to study the applicability of RL to the DAC problem. We compare a standard Q-learning (Watkins, 1989) approach to baseline agents and show that RL can learn general configuration policies when paired with function approximation methods (van Hasselt et al., 2016). We further show that the considered RL methods are more capable of handling various short effective sequence lengths or levels of stochasticity of the reward signal compared to the baselines. Finally, this is the first time we considered self-paced learning in the contextual RL setting and could show that it can improve learning speeds of RL agents.

The problem definition that is discussed in this chapter as well as the proposed solution approach based on contextual reinforcement learning is used throughout this dissertation.

Chapter 6: Automated Dynamic Algorithm Configuration

This chapter provides a comprehensive account of dynamic algorithm configuration and extends the preceding chapter. In particular, this chapter extends the previous one by providing a

- more thorough discussion and classification of related work from different sub-fields of AI;
- discussion of solution methods for dynamic algorithm configuration beyond reinforcement learning;
- description of three case studies using and extending DACBench, confirming DAC as a practical alternative to prior meta-algorithmic frameworks;
- discussion of current limitations of dynamic algorithm configuration and the so far evaluated solution approaches.

Note that chronologically, this chapter was written last during the author's Ph.D. and consolidates multiple previous papers that will appear in later chapters.

Chapter 7: Learning Step-Size Adaptation in CMA-ES

In this chapter, we present an in-depth case study of DAC by reinforcement learning on a real algorithm and problem instances. We make use of the problem definition we defined in the previous chapters to derive the application-specific problem of learning to adapt the step-size in an evolutionary algorithm, in the form of CMA-ES (Hansen, 2006). The step size is continuously variable and typically is adapted while CMA-ES is running through the use of hand-designed heuristics (Hansen and Ostermeier, 2001).

We define all relevant components of the underlying cMDP to be able to make use of reinforcement learning. This includes the state- and action spaces as well as the reward function. Following related work on learning to optimize (LTO; Li and Malik, 2017), we propose to make use of the guided policy search method (GPS; Levine and Abbeel, 2014;

Levine and Koltun, 2013) to learn step-size adaptation policies. For many algorithms, there exist meaningful hand-designed heuristics that can take care of adapting an algorithm's parameters. To make use of such expert knowledge, we present an extension to the GPS method which can repeatedly query the heuristics to improve the learned policy. For our purposes, we use the commonly used CSA heuristic (Hansen and Ostermeier, 2001).

We conduct a large-scale empirical study in which we evaluate the efficacy of our proposed DAC approach. In the experiments, we compare the performance of the learned policies to that of the baseline CSA policy with an optimized initial starting value. We begin by considering the simplest example where we learn a policy for optimization of a single 10D function. The learned DAC policies outperform the hand-crafted baseline. We then consider more complex scenarios where we learn policies that are transferred to (i) functions with higher dimensions, (ii) unseen test functions, and (iii) longer optimization trajectories. For all the considered scenarios we demonstrate that the learned policies are capable of generalizing to these settings while still outperforming the baseline policy.

This case study shows how to tackle DAC in a continuous action space, how to incorporate prior knowledge into DAC policies, and provides ample evidence that the contextual RL setting allows learning policies that are capable of generalizing to unseen settings.

Chapter 8: Learning Heuristic Selection with Dynamic Algorithm Configuration

While the previous case study tackles DAC in a continuous configuration space, in this case study we consider a discrete action space. In this chapter, we propose to learn heuristic selection via DAC for use in AI planning to increase the speed of planning systems. Further, this chapter provides a formal analysis of DAC and shows that it is a generalization of algorithm selection and configuration. We could prove that, for a particular family of AI planning problems, a DAC policy that dynamically selects which heuristic to follow at each step, will require exponentially fewer steps than a configuration determined with previous meta-algorithmic frameworks. Based on this theoretical analysis we propose a set of white-box benchmarks that can be used to validate the performance of DAC policies without having to perform expensive algorithm runs on real problem instances.

To empirically validate DAC in this setting, we propose the necessary components of the cMDP, including the reward function, and the action- and state spaces. We propose to use a variant of the popular Deep Q-Networks (DQN; Mnih et al., 2015), in the form of double deep Q-Networks (DDQN; van Hasselt et al., 2016) as DQNs are particularly suited for discrete action spaces. Further, we propose a general interface that makes use of communication via sockets. This allows us to easily interface a DAC agent, which is typically implemented in python, with algorithms that might be written in a different language. In this specific case study, this allowed us to interface DAC with the FastDownward system (Helmert, 2006) which is implemented in C++.

We perform a large empirical study, using problem instances from a variety of planning domains. The learned policies are very capable, readily outperform the considered baselines, and even outperform the theoretical best (static) algorithm selector, showing the power of DAC policies.

Chapter 9: DACBench: A Benchmark Library for Dynamic Algorithm Configuration

Following the insights gained from the preceding case studies, in this chapter, we propose a benchmark suite for DAC which we dub DACBench. DACBench consolidates all benchmarks designed in the previous chapters as well as includes benchmarks from prior literature. To

facilitate better reproducibility and to lower the barrier of entry we propose a standardized interface, building on the popular OpenAI gym interface (Brockman et al., 2016).

We extensively discuss the desiderata of dynamic configuration benchmarks and how they influenced DACBench. In particular, DACBench focuses on accessibility, reproducibility and aims at facilitating further DAC research. To this end, DACBench contains (i) cheap-to-run artificial benchmarks that do not require running real algorithms, (ii) white-box benchmarks that require interaction with real algorithms but are designed with specific problem characteristics in mind, and (iii) benchmarks using real algorithms and real problem instances. The last category of benchmarks comes from a broad variety of AI domains, consisting of AI planning, evolutionary algorithms, and deep learning. All benchmarks are studied along various degrees of difficulty, such as the degree of randomness or the effect that problem instances have on the benchmark. We conclude this chapter with a discussion on the challenges of dynamic algorithm configuration in practice as well as research.

Chapter 10: Theory-inspired Parameter Control Benchmarks for Dynamic Algorithm Configuration

We propose a novel benchmark for dynamic algorithm configuration inspired by theoretical insights from parameter control research, a closely related research field. Parameter control adapts hyperparameters on-the-fly but not per-instance and is predominantly used in evolutionary computation (see, e. g., B. Doerr and C. Doerr, 2018, for an overview). Parameter control is studied in both theory and practice. Based on the theory research in this setting, it is possible to compute optimal dynamic configuration schedules for different algorithms and classes. For this novel benchmark, we study a parameterized version of the classical randomized local search method $(1 + 1)$ RLS on LEADINGONES. The runtime distribution is very well understood in this setting (B. Doerr, 2019) which can be used to verify the performance of proposed DAC solution approaches. In particular, for this setting, we can compute the exact optimal policies for different problem instance sizes as well as different configuration spaces of the mutation parameter. This provides novel insights into this setting as well as enables us to provide the ground truth for real algorithm runs on real problem instances, which was previously not possible to compute in DACBench. This benchmark thus closes an important gap in the setup of DACBench and increases its usability. Finally, we use this benchmark and demonstrate how it can be used to study the capabilities of a DDQN for learning DAC policies with different action spaces as well as different problem sizes.

Chapter 11: TempoRL: Learning When to Act

We now turn our attention towards improving reinforcement learning methods following insights we gained by making use of it for solving DAC problems. In particular, in this chapter, we discuss an improvement to RL which was inspired by our case study on DAC for AI Planning. In this case study, we observed that learned successful DAC policies often need to repeat the same action for multiple steps, as the solution landscape has not significantly changed. Still, in other situations having the ability to switch configurations at every step is beneficial to adapt to rapid changes. Thus this raises the question, when is it necessary to make a new decision with the reinforcement learning agent? To tackle this question we propose TempoRL, a proactive alternative to the classically more reactive RL paradigm. Further, to learn such proactive strategies efficiently we propose to dynamically configure the temporal exploration for value-based RL.

Inspired by cMDPs we formulate skip MDPs to model the problem of learning temporal exploration. We use this model to derive a hierarchical learning approach that jointly learns which action to play in a given state as well as how long this action should be repeated.

This hierarchical approach lets us jointly learn to optimize the reward signals as well as the temporal-exploration hyperparameter. We demonstrate the effectiveness of TempoRL on a variety of problems, starting from tabular methods on simple grid worlds up to deep RL methods in more challenging environments, such as Atari games (Bellemare et al., 2013).

Chapter 12: Self-Paced Context Evaluation for Contextual Reinforcement Learning

While our work on solving DAC by contextual reinforcement learning in the previous chapters has enabled us to learn general policies across problem instances, we did not discuss on which instances to learn. Our foundational work mostly employed simple round-robin switching between instances. Dedicated curricula could increase learning speed by starting to learn on easier instances before progressing towards more difficult ones. In our first work on DAC we already considered to use self-paced learning (SPL; Kumar et al., 2010). In this chapter, however, we build a dedicated self-paced learning mechanism for value-based contextual reinforcement learning which we dub SPACE.

SPACE makes direct use of the current value function estimates of agents to gauge the learning progress of the agent. If between learning updates there are vast differences in predicted value, SPACE treats the problem instance as more difficult for the agent. Over time, SPACE builds a curriculum using this difficulty estimate, which incorporates more and more difficult problem instances. In this way, SPACE smoothly progresses through the collection of problem instances without exposing the agent to large jumps in difficulty. Further, we prove that SPACEs performance is upper-bounded by the previously used round-robin selection scheme. Our final evaluation of SPACE demonstrates that it readily outperforms the round-robin baseline as well as related self-paced learning methods.

Chapter 13: CARL: A Benchmark for Contextual and Adaptive Reinforcement Learning

Our work on using contextual RL for DAC has proved that it is a powerful approach for generalization in deep reinforcement learning research. However, so far no dedicated contextual RL method exists which might be used for learning general agents. We believe this can in a large part be attributed to the lack of benchmarks dedicated to contextual reinforcement learning. Most existing benchmarks that could be used to assess the generalization of deep RL agents hide the notion of context, implicitly provide it as part of image observations, and procedurally generate problem instances (see, e. g., the survey by Kirk et al. (2021) for an overview on the topic). Such benchmarks make it difficult to academically judge the generalization capabilities of trained agents. To facilitate better studies and development of methods for generalization of (contextual) reinforcement learning agents we present a novel benchmark collection which we dub CARL.

CARL extends existing reinforcement learning benchmarks for contextual learning based on easy-to-understand physical properties, such as changes in friction. These physical properties provide a clear view of what is changing within the environment. While DACBench can also be considered a benchmark for contextual RL, it comes with additional confounding factors and not as easily changeable context features. DACBench context features are dependent on the problem instance and domain and are mostly not as easily interpretable to non-domain experts. Thus CARL allows to more easily study the influence of context on RL agents. We use CARL to assess the in- and out-of-distribution generalization capabilities of commonly used RL agents and study how the context distribution influences the learning dynamics of agents.

Lastly, in this chapter, we provide a discussion on challenges for learning general reinforcement learning agents which can be tackled using CARL and which will feedback

into further DAC research. These challenges include (i) representation learning, (ii) uncertainty of RL agents, (iii) continual learning, (iv) interpretable and explainable deep RL, (v) AutoRL and (vi) high confidence generalization.

2.3 List of Publications

In this section, we list all core publications that build the backbone of this dissertation as well as related work by the author. We provide a detailed description of author contributions for core publication at the beginning of the respective chapters.

Core Publications This dissertation consolidates the following nine research papers. All research papers follow the common scientific question of *how can we train general reinforcement learning agents for dynamic configuration of algorithms*. To facilitate maximal reproducibility of all listed works we open-sourced all our code and provide links to the corresponding code repositories.

- A. Biedenkapp, H. Bozkurt, T. Eimer, F. Hutter, and M. Lindauer (June 2020). "Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework". In: *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*. Ed. by J. Lang, G. De Giacomo, B. Dilkina, and M. Milano, pp. 427–434.
  Code: `https://github.com/automl/DAC`

- G. Shala, A. Biedenkapp, N. Awad, S. Adriaensen, M. Lindauer, and F. Hutter (2020). "Learning Step-Size Adaptation in CMA-ES". in: *Proceedings of the Sixteenth International Conference on Parallel Problem Solving from Nature (PPSN'20)*. Ed. by T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, and H. Trautmann. Lecture Notes in Computer Science. Springer, pp. 691–706.
  Code: `https://github.com/automl/LTO-CMA`

- D. Speck, A. Biedenkapp, F. Hutter, R. Mattmüller, and M. Lindauer (2021). "Learning Heuristic Selection with Dynamic Algorithm Configuration". In: *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21)*. Ed. by H. H. Zhuo, Q. Yang, M. Do, R. Goldman, S. Biundo, and M. Katz. AAAI, pp. 597–605.
  Code: `https://github.com/speckdavid/rl-plan`

- T. Eimer, A. Biedenkapp, M. Reimer, S. Adriaensen, F. Hutter, and M. Lindauer (2021b). "DACBench: A Benchmark Library for Dynamic Algorithm Configuration". In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI'21*. Ed. by Z. Zhou. ijcai.org, pp. 1668–1674.
  Code: `https://github.com/automl/DACBench`

- A. Biedenkapp, R. Rajan, F. Hutter, and M. Lindauer (2021). "TempoRL: Learning When to Act". In: *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 914–924.
  Code: `https://github.com/automl/TempoRL`

- T. Eimer, A. Biedenkapp, F. Hutter, and M. Lindauer (2021a). "Self-Paced Context Evaluation for Contextual Reinforcement Learning". In: *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 2948–2958.
  Code: `https://github.com/automl/SPaCE`

- C. Benjamins, T. Eimer, F. Schubert, A. Biedenkapp, B. Rosenhan, F. Hutter, and M. Lindauer (2021). "CARL: A Benchmark for Contextual and Adaptive Reinforcement Learning". In: *Workshop on Ecological Theory of Reinforcement Learning (EcoRL@NeurIPS'21)*.
  Code: `https://github.com/automl/CARL`

- A. Biedenkapp, N. Dang, M. S. Krejca, F. Hutter, and C. Doerr (2022a). "Theory-inspired Parameter Control Benchmarks for Dynamic Algorithm Configuration". In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'22)*. Ed. by J. Fieldsend. ACM.
  Code: `https://github.com/ndangtt/LeadingOnesDAC`
  Note: *Best paper award GECCO'22 (GECH Track)*.

- S. Adriaensen, A. Biedenkapp, G. Shala, N. Awad, T. Eimer, M. Lindauer, and F. Hutter (2022). "Automated Dynamic Algorithm Configuration". In: *arXiv:2205.13881 [cs.AI]*.
  Code: `https://github.com/automl/2022_JAIR_DAC_experiments`

Further Publications Here, we list further publications by the author that were conducted during their doctoral research. While these are related to the ideas and concepts presented in the following pages, they are out of scope of this dissertation.

The first set of related publications by the author discusses (dynamic) configuration in the realm of automated reinforcement learning (AutoRL):

- J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, F. Hutter, and M. Lindauer (2022). "Automated Reinforcement Learning (AutoRL): A Survey and Open Problems". In: *Journal of Artificial Intelligence Research (JAIR)* 74, pp. 517–568.

- B. Zhang, R. Rajan, L. Pineda, N. Lambert, A. Biedenkapp, K. Chua, F. Hutter, and R. Calandra (2021). "On the Importance of Hyperparameter Optimization for Model-based Reinforcement Learning". In: *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*. ed. by A. Banerjee and K. Fukumizu. Proceedings of Machine Learning Research.

- J. KH Franke, G. Köhler, A. Biedenkapp, and F. Hutter (2021). "Sample-Efficient Automated Deep Reinforcement Learning". In: *Proceedings of the International Conference on Learning Representations (ICLR'21)*. Published online: `iclr.cc`.

The second set of publications discusses configuration and hyperparameter optimization tools and methods:

- M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter (2022). "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization". In: *Journal of Machine Learning Research (JMLR) – MLOSS* 23.54, pp. 1–9.

- S. Izquierdo, J. Guerrero-Viu, S. Hauns, G. Miotto, S. Schrodi, A. Biedenkapp, T. Elsken, D. Deng, M. Lindauer, and F. Hutter (2021). "Bag of Baselines for Multi-objective Joint Neural Architecture Search and Hyperparameter Optimization". In: *Workshop on Automated Machine Learning (AutoML@ICML'21)*.

- N. Awad, G. Shala, D. Deng, N. Mallik, M. Feurer, K. Eggensperger, A. Biedenkapp, D. Vermetten, H. Wang, C. Doerr, M. Lindauer, and F. Hutter (2020). "Squirrel: A Switching Hyperparameter Optimizer Description of the entry by AutoML.org & IOHprofiler to the NeurIPS 2020 BBO challenge". In: *arXiv:2012.08180 [cs.LG]*.

- M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter (2019a). "BOAH: A Tool Suite for Multi-Fidelity Bayesian Optimization & Analysis of Hyperparameters". In: *arXiv:1908.06756 [cs.LG]*. URL: https://arxiv.org/abs/1908.06756.

Lastly, the third set of publications discusses methods and benchmarks to gain a better understanding of optimization procedures and algorithms:

- René Sass, Eddie Bergman, André Biedenkapp, Frank Hutter, and Marius Lindauer (2022). "DeepCAVE: An Interactive Analysis Tool for Automated Machine Learning". In: *Workshop on Adaptive Experimental Design and Active Learning in the Real World (ReALML@ICML'22)*. DOI: 10.48550/arXiv.2206.03493.
- R. Rajan, J. L. B. Diaz, S. Guttikonda, F. Ferreira, A. Biedenkapp, J. O. von Hartz, and Frank Hutter (2020). "MDP Playground: Controlling Dimensions of Hardness in Reinforcement Learning". In: *arXiv:1909.07750*. URL: https://arxiv.org/abs/1909.07750.
- M. Lindauer, M. Feurer, K. Eggensperger, A. Biedenkapp, and F. Hutter (2019b). "Towards Assessing the Impact of Bayesian Optimization's Own Hyperparameters". In: *IJCAI 2019 DSO Workshop*. Ed. by P. De Causmaecker, M. Lombardi, and Y. Zhang.
- A. Biedenkapp, J. Marben, M. Lindauer, and F. Hutter (2018). "CAVE: Configuration Assessment, Visualization and Evaluation". In: *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*. ed. by R. Battiti, M. Brunato, I. Kotsireas, and P. Pardalos. Lecture Notes in Computer Science. Springer.

Further Achievements As part of the team *AutoML.org & IOHprofiler, featuring the switching squirrel* the author contributed to a black-box optimizer that dynamically switches between black-box optimization methods. This proved to be a powerful approach as the team won the meta-learning friendly track of NeurIPS' 2020 black-box optimization challenge.[5] Further, the team would have achieved 3rd place on the main track of the competition[6] if not for a minor bug in the final submission. A description and detailed analysis of the competition is given in (Turner et al., 2021). The description of the approach and the resulting switching optimizer is given in (Awad et al., 2020). Further, a core publication of the dissertation "Theory-inspired Parameter Control Benchmarks for Dynamic Algorithm Configuration" won a best paper award at the *GECCO'22* conference on the *General Evolutionary Computation and Hybrids* track.[7]

[5] https://bbochallenge.com/altleaderboard

[6] https://bbochallenge.com/leaderboard

[7] https://gecco-2022.sigevo.org/Best-Paper-Awards#GECH_Track

CHAPTER 3

Additional Related Work

As this work is a *cumulative dissertation (dissertation by publication)*, every chapter comes with its own related work section. Thus, for an in-depth discussion on dynamic algorithm configuration in the context of meta-algorithmics we refer to Section 6.2. In the following, we extend this discussion with an emphasis on automated machine learning (AutoML; Hutter et al., 2019).

AutoML uses meta-algorithmics to make machine learning (ML) more accessible and reduce the required computational resources. To this end, AutoML considers tuning all elements of a potential ML pipeline. Thus, DAC could be a powerful tool to advance AutoML research. In the following, we discuss a few application areas where learned DAC policies have been successfully applied to AutoML.

There are many potential applications in machine learning where dynamic configuration is needed. The prototypical example concerns learning rates in deep learning. Too high learning rates might lead to diverging learning behavior whereas too small learning rates might lead to very slow to no training (Bach and Moulines, 2011). Typical solution approaches to this problem are most often hand-designed schedules which come with their own parameters that need to be adapted to the problem at hand. With the prominence of this problem in the machine learning community, it is unsurprising that this is one of the first problems in machine learning to which – what we have come to call – DAC was applied to learn dynamic configuration policies. Daniel et al. (2016) proposed to use a reinforcement learning method to learn to adapt learning rates based on observed progress of the neural network training process. The trained policies could perform on par or even outperform the best baseline. Further, the trained policies showed some generalization capabilities by successfully adapting learning rates for slightly larger networks, related datasets, as well as architecture types. Similar approaches and results were reported by C. Xu et al. (2017) and Z. Xu et al. (2019). While these works are only the first steps, similarly pre-trained policies could help to drastically reduce the required compute for training deep neural networks by adapting to the context (e.g., dataset and/or architecture) at hand.

More recently, Almeida et al. (2021) proposed to adapt the hyperparameters of an optimizer, such as Adam (Kingma and Ba, 2015), instead of learning to directly update neural network weights. Counter to the previous line of research the authors tuned not only the learning rate but also parameters affecting weight decay and gradient clipping. To this end, a reinforcement learning method is used to update the optimizer parameters. The resulting learned policies improve the training speed by factors up to 2.5 over the considered baseline and showed promising generalization capabilities by being successfully transferred to larger training tasks.

Still, in the realm of deep learning, DAC has been considered for dynamically configuring ensemble learning with convolutional neural networks (Malashin, 2021). In this setting,

a large pool of potential classifiers is available to build a smaller ensemble. Following the DAC setting, taking context into account about which images to classify and which classifiers are already in the ensemble improved the accuracy over simply returning the average prediction.

Besides these applications in deep learning, there has been work on using DAC in genetic algorithms (Pettinger and Everson, 2002), evolutionary algorithms (Sakurai et al., 2010) and differential evolution (Sharma et al., 2019). The first work used a variant of Q-learning (see the next chapter for details) to adapt the mutation operator on the fly, and the learned policy outperformed a static baseline. Similarly, Sakurai et al. (2010) used Q-learning to learn policies that change the crossover operator on the fly. However, this proposed method was not extensively evaluated and only theoretical considerations on the reward are discussed. Lastly, Sharma et al. (2019) proposed to use Q-learning to adapt the mutation strategy of DE on the fly. The resulting policies improved the performance of DE and showed some transfer capabilities. As these works share similarities, in future work it would be interesting to see if a general DAC policy can be trained such that is is capable of adapting the mutation operator for a variety of different algorithms in the realm of evolutionary computation.

The discussed works clearly show that DAC can be a powerful tool for AutoML. Still, it is not trivially done and more work is needed to incorporate DAC fully in the realm of AutoML. For most parameters, it is not clear if they are better adapted on the fly or fixed at the beginning of the run. Thus, any application in AutoML will require domain knowledge to avoid wasting resources only to figure out that a parameter is best not dynamically adapted. However, of particular interest for this is the emerging field of AutoRL (Parker-Holder et al., 2022). As most variants of reinforcement learning (RL) iteratively change the data by interacting with an environment, RL has a large potential to benefit from DAC research and one can reasonably expect that many parameters will require dynamic configuration. Still, with the large resource requirements of current RL, there has not yet been any approach to learning general configuration policies in a DAC-like fashion. We, however, expect that DAC methods developed in AutoRL will likely also be useful for AutoML.

Chapter 4

Reinforcement Learning

Throughout this dissertation, reinforcement learning (RL) plays a crucial role. RL methods are our chosen go-to solution method for dynamic algorithm configuration problems. In this chapter, we will give the necessary background on RL (for a comprehensive account of RL, and in particular theoretical considerations, we refer to Sutton and Barto (2018)). The subsequent chapters will then discuss RL in the context of DAC and why it is our chosen solution method.

RL is a type of machine learning in which intelligent agents learn by interacting with an environment in a trial-and-error fashion. The interactions generate experiences that inform an agent about the consequences of their actions. This generated experience can then be used to reinforce good behaviors while weakening bad behaviors. Take, for example, a simple agent that is tasked with navigating a grid world as depicted in Figure 4.1. At every time-step the agent can choose to execute one of four actions `Up`, `Down`, `Left` or `Right`. Its goal is to move from the starting state in the bottom left to the goal in the bottom right as fast as possible. To do so successfully, it has to avoid falling down a cliff (depicted by black squares). A single trial stops after a maximum number of steps have been taken, the agent has fallen down the cliff, or it successfully reached the goal. After every trial (and even after every step) an agent can see if it was successful or not and adapt its behavior accordingly. With the updated behavior it can run another trial to further update its behavior. This

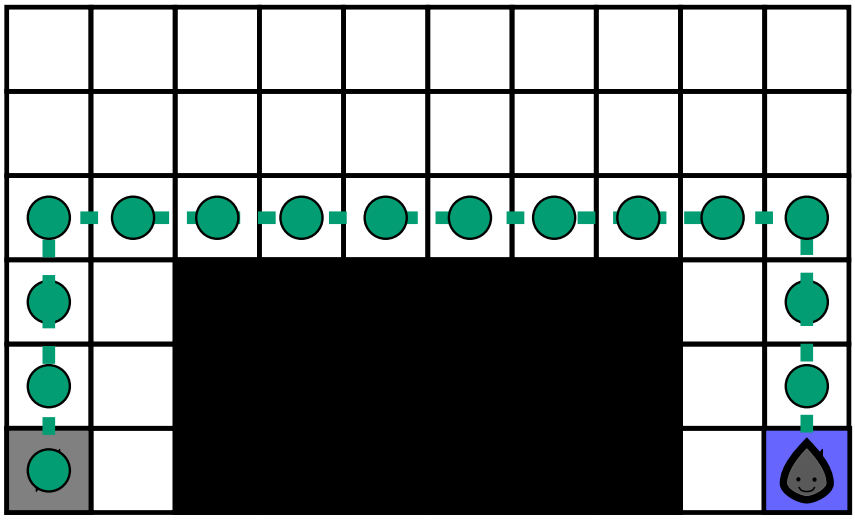


Figure 4.1: Example grid world adapted from our publication (Biedenkapp et al., 2021). Starting from the bottom left, an agent has to reach the goal in the bottom right without falling down a cliff (black squares).

pattern gets repeated until it has learned to perfectly solve the task. In RL, this interaction between a learning agent and environment is modeled as a Markov decision process (MDP; Bellman, 1957).

4.1 Markov Decision Processes

An MDP provides a straightforward formulation of the problem of learning by interaction with the environment. At its core, an MDP is a 4-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ with $\mathcal{S}$ the state space, $\mathcal{A}$ the action space, transition function $\mathcal{T}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and reward function $\mathcal{R}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The transition function, also referred to as transition dynamics, gives the probability of reaching a possible successor state s' when playing an action a in state s. The (potentially stochastic) reward function describes the reward r obtained by playing an action a in state s. Note that typically it is assumed that an MDP fulfills the Markov property, i.e., the transition dynamics only depend on the immediate predecessor state. If that is the case, the state contains all relevant information about past interactions and how they will affect future interactions. For some applications it is convenient to include an initial state distribution $\mathcal{P}$, discounting factor γ, or a set of terminal states $\mathcal{S}^\dagger$ into the MDP formulation though they are typically not directly taken into account by learning agents.[8]

An agent interacts with an environment at discrete[9] time-steps $t \leq T \in \mathbb{N}_0$. Starting out in some initial state $s_{t=0} \sim \mathcal{P}$ the agent decides which action $a_{t=0} \in \mathcal{A}$ to execute. This action advances the environment and the consequences of this are made apparent to the agent at the next time step by receiving a reward signal r_{t+1} and observing the successor state s_{t+1}. Actions need to be chosen such that the cumulative future reward J_t is maximized, with $J_t = r_t + r_{t+1} + \cdots + r_T = \sum_{k=t}^{T} r_k$. As T could potentially be infinite, we typically consider a discounted variant of this sum

$$J_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{4.1}$$

where the discount factor $0 \leq \gamma \leq 1$ determines how to discount future rewards. The lower the discount the lower the influence of potential future rewards. Figure 4.2 depicts this agent-environment interaction.

4.2 Learning the Value of a State

Now that we know how to model the interaction between agents and environments, we can use this formulation to discuss how agents can learn reward-maximizing behavior. There are three main variants of reinforcement learning – *value-based*, *policy search* and *model-based* – that propose different ways of solving MDPs. In this section, we focus on commonly used *value-based* methods. Throughout this dissertation, we predominately used such methods for learning dynamic configuration policies.

As the name suggests, value-based methods are concerned with learning the value of a state. More precisely, they aim at estimating "how good" it is for an agent to be in a certain state based on the observed rewards. To this end, value-based methods concern themselves with learning a *value function*. These can then be used to derive behavior policies directly

[8] In some cases these elements are even treated as tunable parameters of the learning problem. For example, $\mathcal{P}$ was progressively adapted to ever more distant starting positions in a curriculum learning fashion (OpenAI et al., 2019); γ was adapted over time to initially focus on short horizons problems (François-Lavet et al., 2015) and $\mathcal{S}^\dagger$ is typically adapted for lower hierarchies in hierarchical RL approaches (see, e.g., Kulkarni et al., 2016)

[9] For simplicity's sake we only consider the discrete-time setting. We refer to Bertsekas and Tsitsiklis (1996) for discussion of the continuous-time setting.

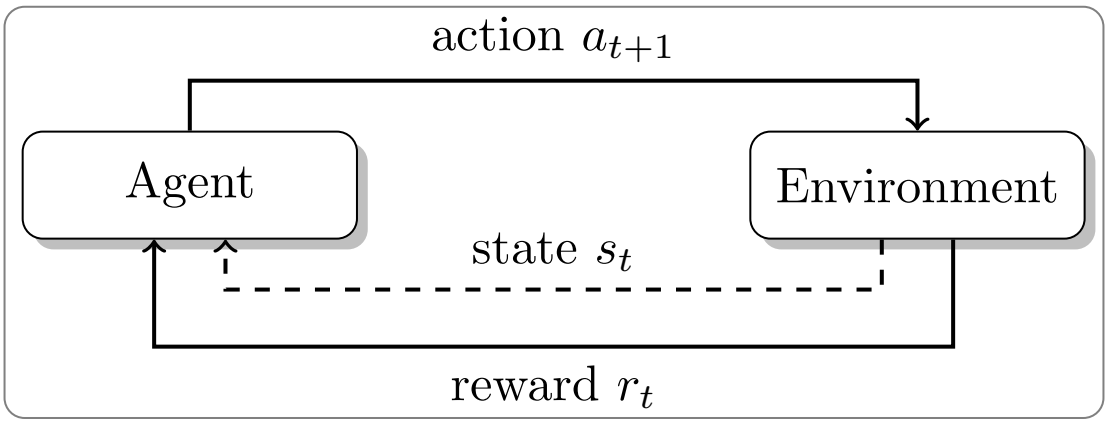


Figure 4.2: Abstract representation of the interaction between agent and environment in a Markov decision process. Image-based on Figure 3.1 of Sutton and Barto (2018).

from the value function. Formally, a policy $\pi: \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is a mapping from states and actions to probabilities that determine how likely it is that action a will be played in state s. *Policy search* methods, on the other hand, directly search in the space of policies to find policies that maximize the cumulative reward. This can, for example, be done by using a neural network as a parameterized policy and iteratively refining the weights, e.g. by means of evolutionary strategies, to improve the reward that can be gained by the policy. Lastly, *model-based* methods aim at learning a model of the underlying transition dynamics of the MDP. This model is then later used to derive policies by creating action plans based on the learned model.

While there are impressive results for both policy search methods (see, e.g., Chrabaszcz et al., 2018) and model-based methods (see, e.g., Hafner et al., 2021), estimating value functions still plays a central role in most reinforcement learning algorithms (Sutton and Barto, 2018). The *state-value function* encodes the expected future cumulative reward when starting in state s at time t and following a policy π from then on as,

$$\mathcal{V}^{\pi}(s) = \mathbb{E}^{\pi}\left[J_t | s_t = s\right] = \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right], \forall s \in \mathcal{S}, \tag{4.2}$$

where J_t is the discounted cumulative future reward as defined in Equation (4.1). Important to note here is that the value in terminal states is always assumed to be zero. Further, the state-value function can be written recursively as

$$\begin{aligned}
\mathcal{V}^{\pi}(s) &= \mathbb{E}^{\pi}\left[r_t + \gamma J_{t+1} | s_t = s\right] && (4.3)\\
&= \mathbb{E}^{\pi}\left[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} | s_t = s\right] && (4.4)\\
&= \mathbb{E}^{\pi}\left[r_t + \gamma \mathcal{V}(s_{t+1}) | s_t = s\right]. && (4.5)
\end{aligned}$$

Equivalently, we can define the *action-value function* as

$$\mathcal{Q}^{\pi}(s, a) = \mathbb{E}^{\pi}\left[r_t + \gamma \mathcal{Q}^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\right], \tag{4.6}$$

which defines the value of taking an action a in state s at time t when following policy π from this point onward and is often referred to as $\mathcal{Q}$-function.

Following from the Bellman optimality equation (Bellman, 1957/2003), once the optimal value function is known, it is straightforward to obtain optimal policies by taking

the action a in state s that maximizes the expected discounted future reward given the optimal value-function, since

$$\mathcal{V}^*(s) = \max_a \mathbb{E}\left[r_t + \gamma \mathcal{V}^*(s_{t+1}) | s_t = s\right], \tag{4.7}$$

$$\mathcal{Q}^*(s,a) = \mathbb{E}\left[r_t + \max_{a'} \gamma \mathcal{Q}^*(s_{t+1}, a') | s_t = s, a_t = a\right]. \tag{4.8}$$

There are well-known approaches that concern themselves with estimating or learning value functions and solving MDPs, such as dynamic programming (Bellman, 1957/2003) or Monte-Carlo methods (see e.g., Chapter 5; Sutton and Barto, 2018). However, dynamic programming requires full knowledge of the underlying transition model, and Monte-Carlo methods require full-length episodes to estimate the value function. Value-based RL methods avoid these problems by learning the value function from temporal differences (TD; Sutton, 1988), i.e. the difference between two temporally consecutive predictions.

One of the most popular RL algorithms, known as $\mathcal{Q}$-learning (Watkins, 1989), which we used throughout this dissertation, uses temporal differences to learn the $\mathcal{Q}$-function, which is in turn used to derive a policy. Using temporal differences, the $\mathcal{Q}$-function can be learned through straightforward error correction. Given a state s and action a, the $\mathcal{Q}$-value can be updated through

$$\mathcal{Q}(s_t, a_t) \leftarrow \mathcal{Q}(s_t, a_t) + \alpha \underbrace{\Big(\overbrace{r_t + \gamma \max \mathcal{Q}(s_{t+1}, \cdot)}^{\text{TD-target}} - \mathcal{Q}(s_t, a_t) \Big)}_{\text{TD-delta}}, \tag{4.9}$$

where γ is the discounting factor and α the learning rate. The TD-target is the observed current reward together with the discounted maximal future reward when following a greedy policy from this point on, i.e., the value we want to predict correctly with $\mathcal{Q}(s_t, a_t)$. The TD-delta is the temporal difference error between the TD-target and the predicted future reward in the current state. In other words, the TD-delta represents how wrong the predicted $\mathcal{Q}$-value was by taking the observed reward into account. The learning rate α then determines how strongly to update the $\mathcal{Q}$-value according to the computed TD-delta. To ensure exploration of the state space during learning, typically an ϵ-greedy policy is used instead of always using a reward maximizing policy $\pi(s) = \arg\max_{a \in \mathcal{A}} \mathcal{Q}(s, \cdot)$. The ϵ determines the probability of replacing an action a_t with a randomly sampled one $a'_t \sim \mathcal{A}$. For $\epsilon > 0$, there is a non-zero probability of visiting every state during learning. Once learning has been completed, however, we can make use of the greedy policy.

In the simplest implementations, the $\mathcal{Q}$-function is represented as a table and function prediction simply consists of table look-ups. For such tabular approaches, Watkins and Dayan (1992) proved the convergence of $\mathcal{Q}$-learning. Modern (deep) reinforcement learning methods make use of sophisticated function approximators to represent value functions, for which, to the best of our knowledge, no convergence proves or guarantees exist so far. Still, Mnih et al. (2015) popularized[10] the use of deep neural networks (NNs) in RL and proposed the DQN algorithm which models the $\mathcal{Q}$-function with NNs and applies $\mathcal{Q}$-learning. In particular, their proposed convolutional neural network (CNN; LeCun et al., 1999) architecture enabled $\mathcal{Q}$-learning on large image based state-spaces. DQN is capable of learning policies for playing Atari games (Bellemare et al., 2013) on a level comparable to human performance.

Many extensions to DQN have been proposed since its introduction[11]. Of relevance for this dissertation is the extension of DQN to use double $\mathcal{Q}$-learning (van Hasselt et al., 2016).

[10] Prior works on using NNs in RL existed though none had as big an impact on the research community. For a survey on the use of NNs for RL up to the inception of DQN we refer to Section 6 of Schmidhuber (2015).

[11] For an evaluation of many extensions and their combination, we refer to Hessel et al. (2018) and Obando-Ceron and Castro (2021).

Using a single Q-function (and by extension single Q-network) to select the maximizing action in the TD-target and to predict the Q-function often leads to instabilities due to overestimation of the Q-values. To mitigate this, van Hasselt et al. (2016) proposed to use a second Q-function by copying the network weights to decouple action selection from value estimation. The first set of weights is used to select the maximizing action. The second set of weights is kept frozen for short periods at a time and used to predict the value of the state-action pair. The benefit of this is two-fold as i) the short freezing periods lead to increased stability in the predictions and ii) the decoupling of action selection from value prediction reduces value overestimation. This extension is known as double deep Q-networks (DDQN) and builds on earlier work by van Hasselt (2010) which first introduced the concept of *double Q-learning* to decouple action selection from value prediction.[12] In most of the following chapters, we make use of either DQN or DDQN when learning dynamic configuration policies.

In the following Part II, we discuss how we can model the problem of dynamic configuration of algorithms as an MDP and motivate our choice of RL as a solution method for DAC. Further, in Part V we discuss shortcomings of RL based from the lens of DAC and propose solutions to fix these problems.

[12] In the tabular case two completely independent Q-functions are learned to mitigate overestimation bias.

Part II

Dynamic Algorithm Configuration: The Problem

CHAPTER 5

Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework

The content of this chapter has been published as:

> A. Biedenkapp, H. Bozkurt, T. Eimer, F. Hutter, and M. Lindauer (June 2020). "Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework". In: *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*. Ed. by J. Lang, G. De Giacomo, B. Dilkina, and M. Milano, pp. 427–434.

Project Idea. The idea of using reinforcement learning for dynamic optimization was proposed by Frank Hutter and Marius Lindauer. André Biedenkapp proposed to model the process of dynamic optimization as contextual MDP, enabling optimization over time and across problem instances.

Implementation and experimentation. André Biedenkapp led the implementation efforts. H. Furkan Bozkurt implemented the initial versions of the artificial benchmarks, which were extended by André Biedenkapp to allow for the principled study of various aspects of generalization of learned policies and robustness to perturbations. André Biedenkapp implemented RL agents and code to perform hyperparameter optimization of those. Theresa Eimer implemented the self-paced learning (SPL) code. André Biedenkapp conducted most experiments with the experiments on SPL conducted by Theresa Eimer.

Paper writing. André Biedenkapp prepared the first draft of the paper. The draft was revised and edited by Frank Hutter and Marius Lindauer and the final paper version was to a large extent written by André Biedenkapp.

24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework

André Biedenkapp[1] and **H. Furkan Bozkurt**[1] and **Theresa Eimer**[3] and **Frank Hutter**[1,2] and **Marius Lindauer**[3]

Abstract. The performance of many algorithms in the fields of hard combinatorial problem solving, machine learning or AI in general depends on parameter tuning. Automated methods have been proposed to alleviate users from the tedious and error-prone task of manually searching for performance-optimized configurations across a set of problem instances. However, there is still a lot of untapped potential through adjusting an algorithm's parameters *online* since different parameter values can be optimal at different stages of the algorithm. Prior work showed that reinforcement learning is an effective approach to learn policies for online adjustments of algorithm parameters in a data-driven way. We extend that approach by formulating the resulting *dynamic algorithm configuration* as a contextual MDP, such that RL not only learns a policy for a single instance, but across a set of instances. To lay the foundation for studying dynamic algorithm configuration with RL in a controlled setting, we propose white-box benchmarks covering major aspects that make dynamic algorithm configuration a hard problem in practice and study the performance of various types of configuration strategies for them. On these white-box benchmarks, we show that (i) RL is a robust candidate for learning configuration policies, outperforming standard parameter optimization approaches, such as classical algorithm configuration; (ii) based on function approximation, RL agents can learn to generalize to new types of instances; and (iii) self-paced learning can substantially improve the performance by selecting a useful sequence of training instances automatically.

1 Introduction

To achieve peak performance of an algorithm, it is often crucial to tune its parameters. Manually searching for performance-optimizing parameter configurations is a complex and error prone task. General algorithm configuration tools [4, 16, 27] free users from the manual search for well-performing parameters. Such tools have been successfully applied to state-of-the-art AI algorithms of various problem domains, such as mixed integer programming [15], AI planning [12], machine learning [35], or propositional satisfiability solving [18]. One drawback of classical algorithm configuration, however, is that it only yields a fixed configuration that is used during the entire run of the optimized algorithm. It does not take into account that most AI algorithms are iterative in nature and thereby ignores that the optimal target parameter configuration may change over time.

[1] University of Freiburg, Germany,
email: {biedenka, bozkurf, fh}@cs.uni-freiburg.de
[2] Bosch Center for Artificial Intelligence, Germany
[3] University of Hannover, Germany, email: lastname@tnt.uni-hannover.de

From the field of adaptive and reactive heuristics, we already know that non-stationary parameter configurations can indeed improve the performance of algorithms substantially. To automatically obtain policies adjusting parameter configurations online, prior work showed that reinforcement learning (RL) can learn those in a data-driven way and thus the performance of a variety of different algorithms can be automatically improved [23, 32, 6, 10, 34].

Extending prior approaches to be applicable across instances, we formalize the problem of learning dynamic configuration policies of an algorithm's parameters across instance sets (in short *dynamic algorithm configuration* or *DAC*) as a contextual Markov decision process (MDP) and apply reinforcement learning (RL) to it. Our formulation of DAC as a contextual MDP allows explicit handling of instances, which we combine with the self-paced learning scheme [22] to focus on subsets of instances, facilitating faster learning of configuration policies. Furthermore, we propose white-box benchmarks explicitly designed to study dynamic algorithm configuration in a principled manner without confounding factors. On these benchmarks, we study the potential and challenges of our approach. Specifically, our contributions are as follows:

1. We formalize the dynamic configuration of algorithm parameters as a contextual MDP, taking instances into account;
2. We propose new and highly flexible white-box benchmarks that allow to study DAC for scenarios involving: (i) budget constraints, (ii) short effective sequences, (iii) noisy rewards, (iv) different degrees of homogeneity of training and testing instances, as well as (v) strong parameter interaction effects;
3. We propose to use self-paced learning to order instances from easy to complex, facilitating faster transfer learning, compared to learning on an unordered set.
4. We are the first to study dynamic algorithm configuration with reinforcement learning in a controlled setting to shed light on its strengths and weaknesses.

2 Related Work

Meta-algorithmic Frameworks The goal of algorithm selection (AS; [31]) is to learn a selection mechanism, that decides which algorithm, out of a finite set of algorithms is most suited to solve a given instance. Algorithm configuration (AC; [17]) however, not only deals with one-dimensional categorical spaces, but with high-dimensional, conditional and mixed categorical/continuous spaces. AC by itself struggles with heterogeneous instance sets (in which different configurations work best for different instances), but it can be combined with AS to search for multiple well-performing configurations and select which of these to apply to new instances [39, 20]. For

each problem instance, even this more general form of per-instance algorithm configuration (PIAC) still uses stationary configurations[4]. However for different AI applications, dynamic configuration can be more powerful than static or stationary ones.

Adaptive Configurations in Practice A prominent example for parameters that need to be dynamically adjusted is the learning rate in deep learning: a static learning rate can lead to sub-optimal training results and training times [30]. To facilitate fast training and convergence, various learning rate schedules or adaptation schemes have been proposed, but only a few are data-driven [10]. Contrary to hand-designed adaptation schemes, a learned one was much less sensitive to initial starting points. Further a learned configuration policy could generalize to new architectures and larger networks.

In the field of EAs, self-adaptive strategies can change parameters on the fly [21, 11]. These methods, however, are often tailored to one individual problem, rely on heuristics and are also only rarely learned in a data-driven fashion [32], making them applicable only to homogeneous instances. A learned (and even a random) dynamic configuration policy that adjusts the mutation strategy in differential evolution has been shown to outperform non-adaptive strategies [34].

Similarly, reactive search [5] uses handcrafted heuristics to adapt an algorithm's parameters online. To adapt heuristics to the task at hand, hyper-reactive search [3] parameterizes these reactive heuristics and applies PIAC. In contrast, we propose to not only learn which heuristic to apply, given an instance, but to learn how to configure online without the need of hand-designed reactive heuristics.

Relation to Learning to Learn The work we present here can be seen as orthogonal to work presented under the heading of learning to learn (L2L; [2, 25, 8]). Both lines of work intend to learn optimal instantiations of algorithms. The goal of a L2L agent is to learn how to traverse a search space and how to directly modify solution candidates. In contrast, a dynamic configurator learns how a specific algorithm behaves in a search space, based on which the optimal algorithm parameters are selected;[5] modifications of solution candidates are still handled by the configured algorithm.

For example when configuring iterative optimization heuristics, DAC learns when to switch between heuristics given the observed behaviour when applying the heuristics. L2L in essence would learn or discover new heuristics and thus would directly output how to traverse through the search space.

By exploiting existing algorithms and only focusing on dynamically configuring their parameters, DAC may be more sample efficient and generalize better than directly learning algorithms entirely from data, while also preserving guarantees that hold for the existing algorithm regardless of its parameter settings.

3 DAC as Contextual MDP

Definition 3.1 (DAC: Dynamic Algorithm Configuration)**.** *Given a parameterized algorithm A with a configuration space Θ, a probability distribution p over instances $\mathcal{I}$ (which correspond to different inputs to A), a state description $s_t \in \mathcal{S}$ of A solving an instance $i \in \mathcal{I}$ at time point t, and a cost metric $c : \Pi \times \mathcal{I} \to \mathbb{R}$ assessing the cost of a* dynamic configuration policy $\pi \in \Pi$ *on instance i (e.g., runtime to solve an instance, cost of a finally returned solution, or the empirical loss of a predictive model) the goal is to obtain a policy $\pi^* : \mathcal{S} \times \mathcal{I} \to \Theta$, that adapts a parameter configuration $\theta \in \Theta$ at time point t, given a state s_t of A solving instance i, by optimizing its cost across a distribution of instances:*

$$\pi^* \in \arg\min_{\pi \in \Pi} \int_{\mathcal{I}} p(i) c(\pi, i) \, \mathrm{d}i \tag{1}$$

Contextual MDP We propose to formulate DAC as a contextual Markov Decision Process (MDP) $\mathcal{M}_{\mathcal{I}} \coloneqq \{\mathcal{M}_i\}_{i \sim \mathcal{I}}$ with $\mathcal{M}_i \coloneqq (\mathcal{S}, \mathcal{A}, \mathcal{T}_i, \mathcal{R}_i)$. The notion of context $\mathcal{I}$ induces multiple MDPs $\mathcal{M}_i$ with shared action and state spaces, but with different transition and reward functions for a given instance i sampled from a distribution $\mathcal{I}$. The MDP $\mathcal{M}_i$ is a 4-tuple, consisting of a state space $\mathcal{S}$ describing the algorithm state, an action space $\mathcal{A}$ changing the algorithm's parameter settings, a probability distribution $\mathcal{T}_i$ of algorithm state transitions, and a reward function $\mathcal{R}_i$ indicating the progress of the algorithm. Algorithms are often tasked with solving varied problem instances from the same, or similar domains. Searching for well-performing parameter settings on only one instance might lead to a strong performance on that individual instance but might not generalize to new instances. In order to facilitate generalization, we therefore explicitly take instance distributions $\mathcal{I}$ as context into account. In the following, we describe in detail how this context $\mathcal{I}$ influences parts of the individual MDPs.

State and Action Spaces At each time-step t, in order to make informed choices about the parameter values to use, the dynamic configurator needs to be informed about the internal state s_t of the dynamically configured algorithm. Many algorithms collect various statistics that are available at each time-step. For example, a SAT solver might track how variable assignments change over time. This information could be used to inform the dynamic configurator about the algorithm's current behaviour.

The theoretically possible state space does not change when switching between instances, and is shared between all MDPs induced by the context. Thus we consider the same state features which will allow us to learn useful relations across instances. To enrich the state space, we could also add instance-specific information, so-called instance features (e.g. problem size), that could allow us to reason across instances, which could be useful in particular for heterogeneous instance sets [24, 33].

Given a state s_t, the dynamic configurator has to decide how to change the value $v \in \mathcal{A}_h$ of a parameter h or directly assign a value to that parameter, out of a range of valid choices. This gives rise to the overall action space $\mathcal{A} = \mathcal{A}_{h_1} \times \mathcal{A}_{h_2} \times \ldots \times \mathcal{A}_{h_n}$ for n parameters. The action space solely depends on the algorithm at hand and is also shared across all MDPs in $\mathcal{M}_{\mathcal{I}}$, similar to the state space.

Transition Function The transition function describes the dynamics of the system at hand. The probability of reaching state s_{t+1} after applying action a_t in state s_t can be expressed as $p(s_{t+1}|a_t, s_t)$. For simple algorithms and a small instance space, it might be possible to derive the transition function directly from the source code of the algorithm. However, we believe that the transition function cannot be explicitly modelled for most interesting algorithms. Nevertheless, even if the dynamics are not modelled, RL can learn how to optimize policies directly from observed transitions.

[4] Static configurations are unchanged throughout the solving process and are not adjusted to new instances. Stationary configurations stay constant throughout the solving process but might adapt to the instance at hand.

[5] We emphasize that we refer to hyperparameters as algorithm parameters. The goal of DAC is not to update weights (sometimes called the parameters) of a neural network directly.

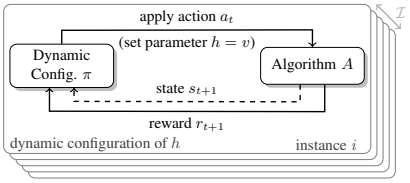


Figure 1: Dynamic configuration of parameter h of an algorithm A on a given instance $i \in \mathcal{I}$, at time-step $t \in T$. Until i is solved or a maximum budget reached, the dynamic configurator decides to change value v of parameter h, based on the internal state s_t of A on the given instance i.

Contrary to the state and action space, the transition function depends on the given instance. For example, an algorithm might be faced with different search landscapes where applying different parameter settings could lead to different state transitions.

Reward Function In order for the dynamic configurator to learn which actions are better suited for a given state, the dynamic configurator receives a reward signal $\mathcal{R}_i(s_t, a_t) \in \mathbb{R}$. Reward functions for DAC include either sparse rewards, e.g., runtime at the end of the algorithm run, or dense rewards, e.g., distance estimations to some goal state or intermediate solution qualities, such as validation error of a partially trained neural network.

As the transition function depends on the instance at hand, so does the reward function. Transitions deemed beneficial by the dynamic configurator on one instance might become unfavorable on another instance, which is reflected by the reward signal.

Interaction of Dynamic Configurator and Algorithm The dynamic configurator's goal is to learn a policy that can be applied to various problem instances i out of a set of instances $\mathcal{I}$, treated as the context of the MDP, see Figure 1. Given an instance i, at time-step t, the dynamic configurator applies action a_t to the algorithm, e.g., setting parameter h to value v. Given this input, the algorithm advances to state s_{t+1} producing a reward signal r_{t+1}, based on which the dynamic configurator will make its next decision. The instance stays fixed throughout the algorithm run.

Learning Policies across Instances Given the MDP and a distribution of instances $\mathcal{I}$, the goal is to find a policy π^* from a space of possible policies Π that performs well across all instances i from a probability distribution $p(i)$ over $\mathcal{I}$. Formally,

$$\mathcal{V}_i^\pi(s_t) = \mathbb{E}\left[r_{t+1}(i) + \gamma \mathcal{V}_i^\pi(s_{t+1}) | s_{t+1} \sim \mathcal{T}_i(s_t, \pi(s_t))\right] \quad (2)$$

$$= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}(i) | s_t = s\right] \quad (3)$$

$$\pi^* \in \arg\max_{\pi \in \Pi} \int_{\mathcal{I}} p(i) \int_{\mathcal{S}_0} \Pr(s_0) \cdot \mathcal{V}_i^\pi(s_0) \, \mathrm{d}s_0 \, \mathrm{d}i \quad (4)$$

$\mathcal{V}_i^\pi$ is the value function, giving the expected discounted future reward, starting from s_t, following policy π (i.e. advancing through $s_{t+1}, s_{t+2}, \ldots$ and adjusting the parameters according to π) on instance i with discounting-rate γ until a termination criterion is met. For finding the optimal configuration policy, we limit ourselves to the set of possible start-states S_0 of the algorithms, which might depend on stochastic initialization or pre-processing of a given instance. In practice, we use simply a Monte-Carlo estimate by performing several runs with different seeds of the algorithm at hand.

Relation to Algorithm Configuration and Selection This formulation of DAC allows to recover classical algorithm configuration (AC) as a special case: in AC, the optimal policy would simply always return the same action, for each state and instance. Further, this formulation also allows to recover per-instance algorithm configuration (PIAC) as a special case: in PIAC, the optimal policy would always return the same action for all states, but potentially different actions across different instances. Finally, algorithm selection (AS) is a special case of PIAC with a 1-dimensional categorical action that merely chooses out of a finite set of algorithms.

Markov Property We argue that most developers in fact already assume the Markov property by using manually designed rule based reactive-heuristics that change parameters if certain conditions are met, independent on how these conditions were met. This behavior satisfies the Markov property since future states (behavior after adaptations) are independent of past states (prior algorithm behavior), given the present (state features).

4 Reinforcement Learning for DAC

Why Reinforcement Learning? In algorithm configuration, a typical black-box optimizer (i) has no access to state information and (ii) sets the parameters only once in the beginning. Both shortcomings hinder classical black-box optimizers from learning optimal sequences of parameters. As a proof of concept, context-oblivious agents [1] take state information into account when selecting which action to play next. This enabled these agents to learn sequences of parameters. However, only a history of previous actions was used and employing a richer state information would enable to learn dynamic policies, that are capable of adapting to the context at hand.

RL is a promising candidate to learn DAC policies in a data driven fashion as we showed how to formulate it as a contextual MDP. It has been demonstrated that RL is capable of generalizing to new tasks given enough examples [9]. Given DAC as an MDP we can sample large numbers of episodes given enough compute resources. For small action and state spaces, RL agents can be easily implemented using table lookups, for large spaces, function approximation methods can make learning feasible. In our experiments, we evaluated ϵ-greedy Q-learning [38] in the tabular setting as well as using function approximation inspired by DQN [29].

Self-Paced Learning for Dynamic Algorithm Configuration Since evaluating a policy on a single instance can already require quite some time (e.g., solving an NP-hard problem), evaluating a policy on all instances is often not feasible in practice. As, shown for classical algorithm configuration, using too few instances likelys result in overfitting and too many instances is too costly [17]. Therefore, we need an efficient, dynamic approach for selecting a subset of instances for training a dynamic configurator.

Similarly to curriculum learning [7], self-paced learning (SPL; [22]) aims to order tasks from easy to complex such that a configurator can transfer knowledge from easier to harder tasks, improving the overall learning. In DAC, these tasks relate to the instances the algorithm has to solve. In contrast to curriculum learning, however, the curriculum is dynamically adjusted to the pace of the learning process. In SPL, the goal is to maximize the reward achievable by a dynamic configurator on the current curriculum by jointly learning the dynamic configuration policy π and the curriculum $\mathbf{v} \in [0, 1]^{|\mathcal{I}|}$

Benchmark Outline 1: Luby

Benchmark Parameters: *minimal episode length L, maximal episode length T, noise level σ*;
$i \sim$ sample_instance;
Actions: $a_t \in \{0, 1, \ldots, \lfloor \log_2 T \rfloor\}$ *for all* $0 \leq t \leq L \leq T$;
States: $s_t \in \{t, \text{Hist}(a_{t-4}, a_{t-3}, \ldots, a_t), i\}$;
for $t \in \{0, 1, \ldots, L\}$ **do**
  $l_t \leftarrow \text{luby}(t, i)$;
  if $a_t \neq l_t$ **then**
    $\text{reward}_t \sim \mathcal{N}(-1, \sigma^2)$;
    $L \leftarrow \min(L + |a_t - l_t|, T)$;
  else $\text{reward}_t \leftarrow 0$;
end

Benchmark Outline 2: Sigmoid

Benchmark Parameters: *number of actions H, number of action values C_h, episode length T*;
$s_i \sim \mathcal{U}(-100, 100, H)$;
$p_i \sim \mathcal{N}(T/2, T/4, H)$;
Actions:
$a_{h,t} \in \left\{ \frac{0}{C_h}, \frac{1}{C_h}, \ldots, \frac{C_h}{C_h} \right\} \forall\, 0 \leq h < H; 0 \leq t \leq T$;
States: $s_t \in s_i \cup p_i \cup \{t\}$;
for $t \in \{0, 1, \ldots, T\}$ **do**
  $\text{reward}_t \leftarrow \prod_{h=0}^{H-1} 1 - abs(sig(t, s_{i,h}, p_{i,h}) - a_{h,t})$;
end

(the i-th element $\mathbf{v}_i$ indicates if instance i belongs to the curriculum):

$$\max_{\pi, \mathbf{v}} \mathcal{C}(\pi, \mathbf{v}, K) = \sum_{i=1}^{|\mathcal{I}|} \mathbf{v}_i \mathcal{R}_i(\pi) - \frac{1}{K} \sum_{i=1}^{|\mathcal{I}|} \mathbf{v}_i \tag{5}$$

where $\mathcal{R}_i(\pi)$ is the reward of following the dynamic configuration policy π on instance i. The term $-\frac{1}{K} \sum_{j=1}^{|\mathcal{I}|} \mathbf{v}_i$ regulates the curriculum size, moving from smaller to larger subsets, given a suitable increasing schedule of K.

Instead of evaluating the dynamic configurator's performance on all instances to determine the true reward $\mathcal{R}_i(\mathbf{w})$, we propose to use the expected reward as given by the $\mathcal{Q}$-function. Easy instances, for which the dynamic configurator already knows well-performing policies, will quickly lead to good rewards which will quickly be reflected in the $\mathcal{Q}$-function. This then lets us efficiently determine which instances should be included in the current curriculum as:

$$\mathbf{v}_i := \begin{cases} 1, & \text{if} \quad \mathcal{C}(\mathbf{w}, \mathbf{v}_i := 0, K) \leq \mathcal{C}(\mathbf{w}, \mathbf{v}_i := 1, K) \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where $\mathbf{v}_i := 0$ excludes the instance in computing the expected reward and $\mathbf{v}_i := 1$ includes it. In each training-iteration we greedily construct the set of training instances from scratch, such that instances are only included if they are expected to improve the reward of the dynamic configurator and then train the dynamic configurator on that set of training instances. If no instance at all is expected to improve the reward, an instance is randomly sampled.

5 White-Box Benchmarks for DAC

As discussed in the related work, various shades of DAC have already been applied to a wide range of AI problems. While they already yielded improved performance in several applications, none of them studied the general DAC problem, and none of them employed a set of carefully-controlled benchmarks with ground truth data to allow a scientific study of when which approaches work well. To remedy this, and to enable an evaluation of DAC policies with full control over all aspects and characteristics of the environment, we propose two highly flexible, white-box benchmarks.

Our benchmarks are designed based on typical challenges in DAC on real algorithms, such as, (i) budget constraints for running an algorithm until a cutoff is reached, (ii) varying lengths of algorithm runs depending on the effectiveness of the chosen parameter settings, (iii) strong parameter interaction effects where the choice of one parameter value influences others, (iv) varying degrees of homogeneity of the instances or (v) noisy rewards because of non-deterministic behavior of algorithms. We focus here on a setting with dense rewards, since in many domains we can approximate the quality of a solution candidate, e.g., validation performance of partially trained deep neural networks or plan quality in optimal AI planning.

Luby To evaluate the ability of agents to dynamically configure algorithms with budget constraints, short effective sequences and noisy rewards across instances of varying degree of heterogeneity, we introduce benchmark *Luby* (see Benchmark Outline 1). The underlying task requires an agent to learn the values in a Luby sequence [28], which is, for example, used for restarting SAT solvers. The sequence is $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, ...$; formally, the t-th value in the sequence can be computed as:

$$l_t = \begin{cases} 2^{k-1} & \text{if } t = 2^k - 1, \\ l_{t-2^{k-1}+1} & \text{if } 2^{k-1} \leq t < 2^k - 1. \end{cases} \tag{7}$$

This gives rise to an action space for sequences of length T with $\mathcal{A} := \{0, 1, \ldots, \lfloor \log_2 T \rfloor\}$ for all time-steps $t \leq T$, with the action values giving the exponents used in the Luby sequence. State information includes the time-step t, the history of actions, and an instance feature describing how the original Luby sequence is shifted.[6]

Inspired by running real algorithms, this benchmark simulates different execution times that depend on the quality of the used policy. The short horizon L (which we dub *short effective sequences*) refers to the minimal time required to solve an instance and thus determines the minimal number of configuration steps. The long term horizon T is equivalent to the cutoff (the maximal time a user wants to run an algorithm) and thus limits the total number of steps. For real algorithm runs, suboptimal parameter settings can lead to longer execution times. To reflect that, in our benchmark, L is increased by the severity of each suboptimal choice, i.e. $L \leftarrow \min(L + |a_t - l_t|, T)$.

Since most algorithms in AI are non-deterministic and do not provide a reliable reward signal, the benchmark uses a fuzzy reward $\mathrm{N}(-1, \sigma^2)$ to penalize wrong action choices, i.e. not the true Luby value l_t at time-step t, in a stochastic way. E.g., setting σ^2 to 1.5 results in a reward where roughly $\frac{3}{4}$ of wrong action choices are correctly penalized and the rest return a false positive signal.

Finally, we task the agent to learn across a distribution of instances. To generate homogeneous instances, every m-th element of the sequence either skips or repeats an element of the true Luby sequence, leading to largely overlapping instances. To generate heterogeneous instances, we sample different starting points of the Luby sequence, leading to little overlap in the resulting instances. For details of the sampling strategies, we refer to the appendix.

[6] $\text{luby}(t, i = 0)$ from Benchmark Outline 1 is given in Equation 7. See appendix for details on $i \neq 0$.

Sigmoid Our second benchmark Sigmoid (see Benchmark Outline 2) allows to study DAC across instance distributions for a varying number of parameters (determined by the scalar H) and varying number of choices per parameter h (determined by C_h). Policies depend on the sampled instance i, which is described by independent sigmoid functions $sig(t; s_{i,h}, p_{i,h}) = \frac{1}{1+e^{-s_{i,h}\cdot(t-p_{i,h})}}$, each of which can be characterized through its inflection points $p_{i,h}$ and scaling factors $s_{i,h}$. The state consists of a time feature, as well as the instance information $s_{i,h}$ and $p_{i,h}$ for each parameter dimension h.

In order to be successful, for each parameter dimension h, an agent has to approximate the sigmoid $sig(t; s_{i,h}, p_{i,h})$ at each time-step t and choose the action $a_{h,t}$ closest to it. For example, for a single parameter ($H = 1$) with only two action values $a_{0,t} \in \{0, 1\}$, an agent would need to learn which value to play first and when to switch to the other value (a concrete example is given in the appendix).

In a multi-parameter setting (i.e. $H > 1$) an agent not only has to learn a simple policy switching between two actions but to learn to follow the shape of each sigmoid function that describe the instance at hand. To simulate interaction effects of the individual parameters, the reward is computed as the product of the individual approximation errors, i.e. $\text{reward}_t \leftarrow \prod_{h=0}^{H-1} 1 - abs(sig(t, s_{i,h}, p_{i,h}) - a_{h,t})$. Further, the granularity of the discretization of the action space can be adjusted by C_h, such that an agent can follow the sigmoid more or less closely, directly affecting its reward.

6 Baselines

As the simplest baseline, we present the best static policy; this defines an upper bound to the performance that could be reached by static algorithm configuration methods, such as SMAC [16]. Our agents, in contrast, can find non-stationary policies that outperform (even optimal) static choices. One could also use SMAC to learn such non-stationary policies by searching for an optimal *sequence* of parameter values (which we dub *parameter scheduling SMAC*, short PS-SMAC). For each time-step, PS-SMAC sets parameter values, making the problem exponentially harder when increasing the episode length. This relates to an optimized schedule of static parameter configurations which ignores all instance features and state information, similar to previous approaches, such as aspeed [14].

As second baseline we consider *context-oblivious agents* [1]. As state information they only take a history of actions into account. During training the agents keep track of the number of times an action lead from one state to another, as well as the average reward this transition produced. This tabular approach limits the agents to small state and action spaces. In our experiments we include *URS*, which selects an action uniformly at random during the training phase; and during the evaluation phase, URS greedily selects the best action given the observations recorded during training. We did not evaluate additional context-oblivious agents due to their limitations on our challenging benchmarks, see the appendix for details.

7 Experimental Study

Setup We used SMAC [16, 26] as a state-of-the-art algorithm configurator and black-box optimizer. We implemented URS using simple tabular 1-greedy Q-learning. Q-learning based approaches (such as URS and our RL-agents) were evaluated using a discounting factor of 0.99 and a constant learning rate of 1.0. The ϵ-greedy agent was trained using a constant $\epsilon = 0.1$. To facilitate generalization to unseen test instances, we include Q-learning using function approximation in the form of a double DQN [37] implemented in chainer [36].[7]

In each training iteration (10^5 in total) each agent observed a full episode. Training runs for all methods were repeated 25 times using different random seeds and each agent was evaluated after updating its policy. When evaluating on the benchmarks we performed 10 evaluation runs of which we report the mean reward. When using a fixed instance set of size 100 on *Sigmoid* we evaluated the agents once on each instance. To allow the tabular Q-learning approaches to work on this continuous state-space we round the scaling factor and inflection point to the closest integer values. We provide further details and results in the supplementary material.[8]

Effect of Short Effective Sequence Length On the *Luby* benchmark with a fixed noise level, we first study the effect of changing the short effective sequence length L, i.e., the minimal sequence length to solve an instance, see Table 1. By construction of the Luby sequence, the optimal static policy is to play the most frequent element in the sequence, i.e. the lowest value, as it makes up roughly 50% of the sequence. With increasing length of the short effective sequence, the reward achievable by this simple policy quickly approaches this 50% threshold (i.e. a reward of 0.5). Given its random behaviour, URS is only able to learn a random policy, which performs much worse than the optimal static policy. Increasing the short effective sequence length degrades PS-SMAC's result as it is only able to find a local optimum, i.e. SMAC identifies that action 0 needs to be played often but not *when* it should be played. Therefore PS-SMAC is unable to outperform the simple static policy. Contrary to the results of PS-SMAC, our ϵ-greedy RL agent is able to adjust its policies better to the presented instance, regardless of the effective sequence length, consistently achieving the best anytime and final reward, readily outperforming the best static policy. However, the greater the intended short effective sequence length, the longer it takes the ϵ-greedy agent to learn (see the appendix for details).

Stochasticity of Reward Signal To study the impact of the stochasticity, we evaluated the agents with different noise levels of the reward and a fixed short effective sequence length, see Table 2. Given very low noise-levels, URS achieves slightly better any-time performance than purely random policies, but still is far off the best static policy. With increasing noise-level, however, URS quickly degrades to a random policy. Due to its black-box nature, PS-SMAC is less affected by the noise coming from a symmetric Gaussian; since it optimizes the cumulative reward of the sequence, the noise is nearly averaged out. In contrast, the RL agent learns to average out the noise for each individual state transition. As a result, our ϵ-greedy agent is hardly more influenced by the noise-level than PS-SMAC, with a drop in AUC by 0.20 compared to PS-SMAC's drop to 0.16.

Homogeneity of Instances The observations made above hold both for more homogeneous and more heterogeneous instance distributions, see Table 1a and 1b as well as Table 2a and 2b. Only PS-SMAC is affected by the change of instance distributions; this is expected since SMAC uses a racing algorithm that assumes a certain degree of homogeneity and given its static algorithm configuration view, PS-SMAC cannot return instance-specific configurations.

[7] We expect that proper tuning of these hyperparameters would further improve the performance of the RL agents, but would be fairly expensive in a real application of DAC.

[8] Appendix and code: `https://github.com/automl/DAC`

	8	16	32
ϵ-greedy	**0.86** (**0.93**)	**0.72** (**0.82**)	**0.47** (**0.65**)
PS-SMAC	0.62 (0.72)	0.39 (0.40)	0.39 (0.40)
URS	0.17 (0.17)	0.17 (0.17)	0.17 (0.17)

(a) Homogeneous

	8	16	32
ϵ-greedy	**0.89** (**0.96**)	**0.75** (**0.84**)	**0.47** (**0.66**)
PS-SMAC	0.56 (0.69)	0.37 (0.39)	0.37 (0.39)
URS	0.17 (0.17)	0.17 (0.17)	0.17 (0.17)

(b) Heterogeneous

Table 1: Results on *Luby* with fuzzy rewards for $L \in \{8, 16, 32\}$ with $T = 64$ on two instance distributions and a noise factor leading to roughly 15% of the actions returning a false positive reward. The values represent the normalized area under the learning curve for 10^5 training episodes. A random policy would achieve 0.17 and the optimal one 1.0. The normalized final performance is given in brackets. The best achieved rewards are highlighted in bold. Respectively, the performance (on both sets) of the best static policy are 0.88, 0.59 and 0.52.

	$p(r_t > 0)$				
	0.01	0.08	0.15	0.20	0.25
ϵ-greedy	**0.96**	**0.92**	**0.86**	**0.81**	**0.76**
PS-SMAC	0.71	0.63	0.62	0.62	0.55
URS	0.21	0.18	0.17	0.17	0.16

(a) Homogeneous

	$p(r_t > 0)$				
	0.01	0.08	0.15	0.20	0.25
ϵ-greedy	**0.97**	**0.94**	**0.89**	**0.84**	**0.80**
PS-SMAC	0.60	0.63	0.56	0.61	0.52
URS	0.21	0.19	0.17	0.17	0.16

(b) Heterogeneous

Table 2: Sensitivity analysis of the presented agents for varying degrees of noise on *Luby*. The short effective sequence was set to 8 with a cutoff of 64. The values represent the normalized area under the learning curve for 10^5 training episodes. The corresponding standard errors and plots are contained in the supplementary material. The first columns in Table 1 correspond to the third columns here.

This effect is amplified in the experiments on *Sigmoid* where PS-SMAC cannot find a policy better than random (see Figure 2 and 3), since it does not take instance features into account and thus cannot distinguish between a positive and negative slope of the sigmoid. Roughly half the instances need completely orthogonal policies to be solved optimally, as the scaling factor is uniformly sampled.

Generalization We study the ability of generalization to unseen instances on the *Sigmoid* benchmark with a single parameter. We note that we evaluated our RL agent not only based on tabular ϵ-greedy, but also based on DQN as we expect function approximation to be crucial for generalization. For this benchmark, the best static policy is to play the action value that is closest to 0.5, as it results in the smallest approximation error on average. This is due to the sampling of the scaling factor and inflection point, where the scaling factor is uniformly sampled between -100 and 100 with the inflection point being normally distributed with a mean at $\frac{T}{2}$. In the binary case both action values are equally preferable. Learning on a distribution of instances (see Figure 2a), DQN learns faster than either tabular approaches and is able to learn an instance-dependent optimal policy, whereas the tabular ϵ-greedy agent gets stuck in a local optimum[9]. Being completely exploratory, URS does not suffer from this problem and recovers the optimal policy. The optimal static policy and PS-SMAC are unable to adapt to the task at hand, resulting in the same reward as a random non-stationary policy.

On the fixed training set (see Figure 2b), results are very similar to the case of learning on a distribution of instances (see Figure 2a); the exception are the tabular agents (URS and ϵ-greedy), which learn much faster (since the possible state-space is much smaller), but which are not able to recover the optimal policy and end up in a local optimum. Furthermore, on the test instances, these tabular agents are incapable of generalization (see Figure 2c), whereas DQN, using function approximation, is able to generalize. Our DQN can quickly generalize from observations on the training set to those on the test instances, resulting in a performance on the test set that only slightly lacks behind the performance on the training instances.

[9] A tuned epsilon schedule might mitigate this problem.

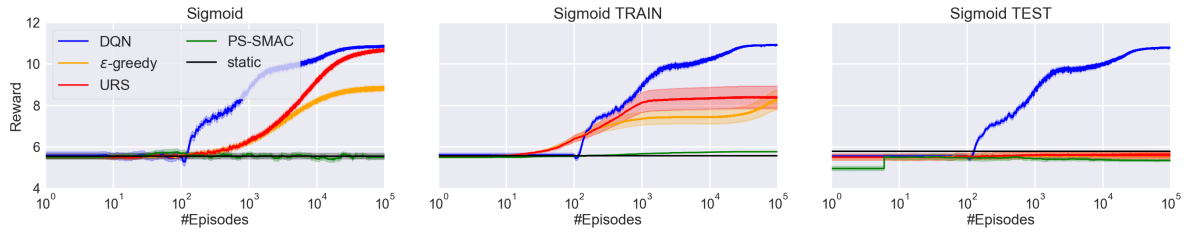
Scaling with the Number of Parameters To study the ability of agents to dynamically configure multiple, strongly interacting parameters, we evaluated them for an increasing number of parameters on the *Sigmoid* benchmark, see Figure 3.

PS-SMAC slowly approaches the same performance as the optimal static policy. For an action space of size 3, PS-SMAC and the static policy are able to achieve a better reward than a random policy. With increasing dimensionality, the static policy outperforms both tabular-agents. Our DQN agent is capable of learning instance-dependent policies even on moderately higher dimensional action spaces. With strong parameter interactions, (see reward of Benchmark Outline 2) learning policies for multiple parameters across a distribution of instances quickly becomes challenging. However, even on the highest presented dimensionality, our DQN is able to outperform the best static policy, while still improving at the end of training. Without longer training nor tuning of the agents' parameters, configuration of five very strongly coupled parameters proves very difficult for the presented agents. Parameter interactions are quite severe, as incrementing the number of parameters roughly halves the reward achievable by a random policy. If one parameter is adjusted suboptimally, this can drastically, negatively influence the overall achievable reward. All agents struggle to cope with such strong interaction effects. Our DQN agent scales best with the number of parameters, as tabular agents cannot model interaction effects.

Effect of Self-Paced Learning We study the effect of using SPL to present a learning agent with new instances ordered from easy to hard and compare it to a simple round-robin (RR) scheme, see Figure 4. SPL first performs poorly but then learns to transfer its learned policies to larger sets of instances. Compared to RR, this substantially improves the final reward, approaching the optimal reward.

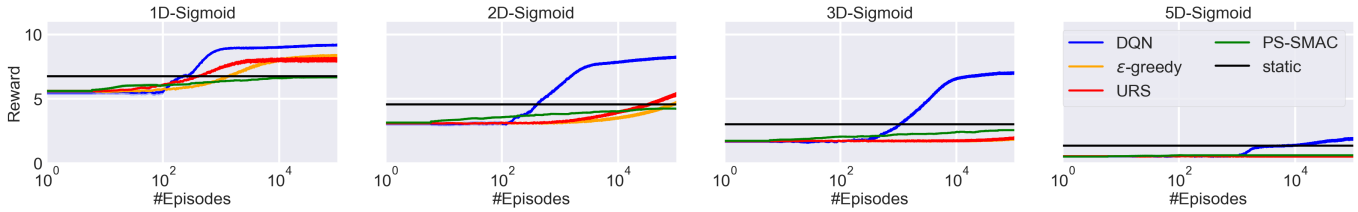
8 Discussion

In practice, the feasibility of RL for DAC depends on several factors. First of all, computing state information and querying the policy to make a decision will induce some overhead. In scenarios with



(a) Sampling from distribution (b) Fixed Training Set (c) Unseen Test Instances

Figure 2: Comparison of generalization to new instances on 1D-*Sigmoid* with binary action space and $T = 11$. The solid line represents the mean reward and the shaded area the standard error over 25 repetitions. To estimate the performance over the distribution of instances in (a), we sample 10 new random sigmoid functions for evaluation. In (b) we evaluate the agents on 100 training instances. In (c) we show the generalization capability by evaluating the agents on 100 new, prior unseen test instances, evaluating them after every training-step in (b) without additional training. A random policy could only achieve a reward of 5.5. The performance of the optimal static policy is given in black.



(a) 1D (b) 2D (c) 3D (d) 5D

Figure 3: Comparison on higher dimensional dynamic configuration problems on $\{1, 2, 3, 5\}$*D-Sigmoid* with $|a_{h,t}| = 3$ and $T = 10$. The solid line is the average performance and the shaded area the standard error over 25 repeated experiments. Due to the interaction effects of the parameters the reward for random policies is halved when incrementing the number of parameters.

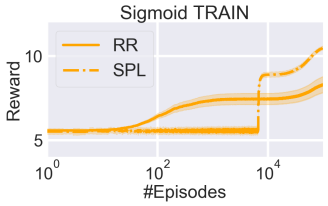


Figure 4: Comparison of training rewards for the ϵ-greedy agent using a round robin (RR) scheme against the same agent using self-paced learning (SPL) on a 1D-Sigmoid with binary actions and $T = 11$.

runtime as a performance metric, it will therefore be of importance to find a good trade-off between the granularity of making decisions and minimizing the overhead. In future work, we plan to jointly learn the optimal parameter value as well as when to adjust the parameter value using recent advances in hierarchical RL.

Furthermore, it is important to have informative state features based on which a policy can change parameter configurations. This is a known problem for RL in general. However, we argue that most AI algorithms anyway collect information for reactive heuristics which could also be used as state information in DAC. Regarding context information, there exists a plethora of work on descriptive instance features, e.g. for AI-planning [13], mixed integer programming [20, 19] or propositional satisfiability solving [40], which can be used for configuration of algorithms from their respective domains.

As always with RL, the reward function is crucial for learning a correct behavior. If an algorithm is able to approximate the quality of solution candidates well, this can be directly used as a reward signal. However, for some algorithms, the quality of solution candidates is hard to approximate and in some domains, runtime-related performance metrics are relevant, e.g., in SAT solving, which cannot be easily approximated ahead of time. Nevertheless even for SAT solving, proxy reward functions were proposed [6] which led to well-performing SAT solvers. Therefore, we believe it viable in future work to carefully design reward functions for many AI domains.

9 Conclusion

We proposed a general framework that enables us to learn configuration policies across instances. To the best of our knowledge we are the first to formalize the dynamic algorithm configuration problem as a contextual MDP, explicitly taking problem instances into account. To study different agent types for the problem of DAC in a controlled setting, we introduced new white-box benchmarks, which enabled us to study DAC with a variety of different properties.

Using these white-box benchmarks, we demonstrated the robustness of using RL for DAC in scenarios with budget constraints, short effective sequences, noisy rewards and demonstrate the ability of RL to handle not only homogeneous but also heterogeneous instances, readily outperforming classical algorithm configuration. We showed the effectiveness of function approximation to handle more challenging state and configuration spaces. We explored the open issue of handling high-dimensional strong parameter interaction effects, where out-of-the box RL methods struggled to scale to higher dimensions. Finally we showed the efficacy of self-paced learning for dynamic algorithm configuration, ordering instances from easy to hard to facilitate faster transfer across instances.

ACKNOWLEDGEMENTS

The authors acknowledge funding by the Robert Bosch GmbH, support by the state of Baden-Württemberg through bwHPC and the German Research Foundation through INST 39/963-1 FUGG.

REFERENCES

[1] S. Adriaensen and A. Nowé, 'Towards a white box approach to automated algorithm design.', in *Proc. of IJCAI'16*, pp. 554–560, (2016).

[2] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, 'Learning to learn by gradient descent by gradient descent', in *Proc. of NeurIPS'16*, pp. 3981–3989, (2016).

[3] C. Ansótegui, J. Pon, M. Sellmann, and K. Tierney, 'Reactive dialectic search portfolios for maxsat', in *Proc. of AAAI'17*, (2017).

[4] C. Ansótegui, M. Sellmann, and K. Tierney, 'A gender-based genetic algorithm for the automatic configuration of algorithms', in *Proc. of CP'09*, pp. 142–157, (2009).

[5] R. Battiti, M. Brunato, and F. Mascia, *Reactive search and intelligent optimization*, volume 45, Springer Science & Business Media, 2008.

[6] R. Battiti and P. Campigotto, 'An investigation of reinforcement learning for reactive search optimization', in *Autonomous Search*, 131–160, Springer, (2011).

[7] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, 'Curriculum learning', in *Proc. of ICML'09*, pp. 41–48, (2009).

[8] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. De Freitas, 'Learning to learn without gradient descent by gradient descent', in *Proc. of ICML'17*, pp. 748–756, (2017).

[9] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, 'Quantifying generalization in reinforcement learning', in *Proc. of ICML'19*, pp. 1282–1289, (2019).

[10] C. Daniel, J. Taylor, and S. Nowozin, 'Learning step size controllers for robust neural network training', in *Proc. of AAAI'16*, (2016).

[11] B. Doerr and C. Doerr, 'Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices', **arXiv:1804.05650**, (2018).

[12] C. Fawcett, M. Helmert, H. Hoos, E. Karpas, G. Roger, and J. Seipp, 'Fd-autotune: Domain-specific configuration using fast-downward', in *Proc. of ICAPS'11*, (2011).

[13] C. Fawcett, M. Vallati, F. Hutter, J. Hoffmann, H. Hoos, and K. Leyton-Brown, 'Improved features for runtime prediction of domain-independent planners', in *Proc. of ICAPS'14*, pp. 355–359, (2014).

[14] H. Hoos, R. Kaminski, M. Lindauer, and T. Schaub, 'aspeed: Solver scheduling via answer set programming', *TPLP*, **15**, 117–142, (2015).

[15] F. Hutter, H. Hoos, and K. Leyton-Brown, 'Automated configuration of mixed integer programming solvers', in *Proc. of CPAIOR'10*, pp. 186–202, (2010).

[16] F. Hutter, H. Hoos, and K. Leyton-Brown, 'Sequential model-based optimization for general algorithm configuration', in *Proc. of LION'11*, pp. 507–523, (2011).

[17] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle, 'ParamILS: An automatic algorithm configuration framework', *JAIR*, **36**, 267–306, (2009).

[18] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. Hoos, and K. Leyton-Brown, 'The configurable SAT solver challenge (CSSC)', *AIJ*, **243**, 1–25, (2017).

[19] F. Hutter, L. Xu, H. Hoos, and K. Leyton-Brown, 'Algorithm runtime prediction: Methods and evaluation', *AIJ*, **206**, 79–111, (2014).

[20] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, 'ISAC - instance-specific algorithm configuration', in *Proc. of ECAI'10*, pp. 751–756, (2010).

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

[22] M. P. Kumar, B. Packer, and D. Koller, 'Self-paced learning for latent variable models', in *Proc. of NeurIPS'10*, pp. 1189–1197, (2010).

[23] M. G. Lagoudakis and M. L. Littman, 'Learning to select branching rules in the DPLL procedure for satisfiability', *Electronic Notes in Discrete Mathematics*, **9**, 344–359, (2001).

[24] K. Leyton-Brown, E. Nudelman, and Y. Shoham, 'Empirical hardness models: Methodology and a case study on combinatorial auctions', *Journal of ACM*, **56**(4), 1–52, (2009).

[25] K. Li and J. Malik, 'Learning to optimize', in *Proc. of ICLR'17*, (2017).

[26] M. Lindauer, K. Eggensperger, M. Feurer, S. Falkner, A. Biedenkapp, and F. Hutter. SMAC v3: Algorithm configuration in Python. `https://github.com/automl/SMAC3`, 2017.

[27] M. López-Ibáñez, J. Dubois-Lacoste, L. Perez Caceres, M. Birattari, and T. Stützle, 'The irace package: Iterated racing for automatic algorithm configuration', *Operations Research Perspectives*, **3**, 43–58, (2016).

[28] M. Luby, A. Sinclair, and D. Zuckerman, 'Optimal speedup of las vegas algorithms', *Information Processing Letters*, **47**(4), 173–180, (1993).

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, 'Human-level control through deep reinforcement learning', *Nature*, **518**(7540), 529–533, (2015).

[30] E. Moulines and F. R. Bach, 'Non-asymptotic analysis of stochastic approximation algorithms for machine learning', in *Proc. of NeurIPS'11*, pp. 451–459, (2011).

[31] J. Rice, 'The algorithm selection problem', *Advances in Computers*, **15**, 65–118, (1976).

[32] Y. Sakurai, K. Takada, T. Kawabe, and S. Tsuruta, 'A method to control parameters of evolutionary algorithms by using reinforcement learning', in *Proc. of SITIS*, pp. 74–79, (2010).

[33] M. Schneider and H. Hoos, 'Quantifying homogeneity of instance sets for algorithm configuration', in *Proc. of LION'12*, pp. 190–204, (2012).

[34] M. Sharma, A. Komninos, M. López-Ibáñez, and D. Kazakov, 'Deep reinforcement learning based parameter control in differential evolution', in *Proc. of GECCO'19*, pp. 709–717, (2019).

[35] J. Snoek, H. Larochelle, and R. Adams, 'Practical Bayesian optimization of machine learning algorithms', in *Proc. of NeurIPS'12*, pp. 2960–2968, (2012).

[36] S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa, S. Saito, S. Suzuki, K. Uenishi, B. Vogel, and H. V. Yamazaki, 'Chainer: A deep learning framework for accelerating the research cycle', in *Proc. of KDD'19*, pp. 2002–2011, (2019).

[37] H. van Hasselt, A. Guez, and D. Silver, 'Deep reinforcement learning with double q-learning', in *Proc. of AAAI'16*, pp. 2094–2100, (2016).

[38] C. Watkins and P. Dayan, 'Q-learning', *Machine learning*, **8**(3-4), 279–292, (1992).

[39] L. Xu, H. Hoos, and K. Leyton-Brown, 'Hydra: Automatically configuring algorithms for portfolio-based selection', in *Proc. of AAAI'10*, pp. 210–216, (2010).

[40] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown, 'SATzilla: Portfolio-based algorithm selection for SAT', *JAIR*, **32**, 565–606, (2008).

CHAPTER 6

Automated Dynamic Algorithm Configuration

The content of this chapter has been published as:

> S. Adriaensen, A. Biedenkapp, G. Shala, N. Awad, T. Eimer, M. Lindauer, and F. Hutter (2022). "Automated Dynamic Algorithm Configuration". In: *arXiv:2205.13881 [cs.AI]*.

Project Idea. The comprehensive journal paper was proposed by Marius Lindauer and Frank Hutter. Steven Adriaensen proposed the discussion of solution approaches beyond reinforcement learning. Based on his ICAPS 2020 keynote, Frank Hutter proposed the formalizations of AC, AS, PIAC, and DAC in very similar terms that directly show DAC's subsumption of the others.

Implementation and experimentation. Implementation and experimentation were jointly carried out by Steven Adriaensen, André Biedenkapp and Gresa Shala with support by Theresa Eimer. Steven Adriaensen implemented and evaluated methods on the SGD benchmark. Gresa Shala implemented and evaluated methods on the CMA-ES benchmark. André Biedenkapp implemented and evaluated methods on the FastDownward benchmark. André Biedenkapp further implemented the code necessary to evaluate the previous meta-algorithmic frameworks in the form of SMACv3 and Hydra.

Paper writing. Paper writing was led by Steven Adriaensen. André Biedenkapp contributed to various sections in part (including related work) and wrote Section 6.2[13] in full. Noor Awad contributed to the related work section. Theresa Eimer wrote Section 5 and Gresa Shala wrote Section 6.1. Marius Lindauer and Frank Hutter revised and edited the final version of the paper.

[13] Section numbering refers to the numbering of the original publication.

Automated Dynamic Algorithm Configuration

Steven Adriaensen adriaens@cs.uni-freiburg.de
André Biedenkapp biedenka@cs.uni-freiburg.de
Gresa Shala shalag@cs.uni-freiburg.de
Noor Awad awad@cs.uni-freiburg.de
University of Freiburg, Machine Learning Lab

Theresa Eimer eimer@tnt.uni-hannover.de
Marius Lindauer lindauer@tnt.uni-hannover.de
Leibniz University Hannover, Institute for Information Processing

Frank Hutter fh@cs.uni-freiburg.de
University of Freiburg, Machine Learning Lab & Bosch Center for Artificial Intelligence

Abstract

The performance of an algorithm often critically depends on its parameter configuration. While a variety of automated algorithm configuration methods have been proposed to relieve users from the tedious and error-prone task of manually tuning parameters, there is still a lot of untapped potential as the learned configuration is *static*, i.e., parameter settings remain fixed throughout the run. However, it has been shown that some algorithm parameters are best adjusted *dynamically* during execution, e.g., to adapt to the current part of the optimization landscape. Thus far, this is most commonly achieved through hand-crafted heuristics. A promising recent alternative is to automatically *learn* such dynamic parameter adaptation policies from data. In this article, we give the first comprehensive account of this new field of *automated dynamic algorithm configuration (DAC)*, present a series of recent advances, and provide a solid foundation for future research in this field. Specifically, we (i) situate DAC in the broader historical context of AI research; (ii) formalize DAC as a computational problem; (iii) identify the methods used in prior-art to tackle this problem; and (iv) conduct empirical case studies for using DAC in evolutionary optimization, AI planning, and machine learning.

1. Introduction

Designing robust, state-of-the-art algorithms requires careful design of multiple components. It is infeasible to know how these components will interact for all possible applications. This is particularly true in the field of artificial intelligence (AI), pursuing ever more general problem-solving methods. This generality necessarily comes at the cost of an increased uncertainty about the problem instances the algorithm will have to solve in practice. To account for this uncertainty, it is common practice to expose difficult design choices as parameters of the algorithm, allowing users to customize them to their specific use case. These algorithm parameters can be numerical (e.g., crossover rate or population size in evolutionary algorithms, and the learning rate or batch size in deep learning), but also categorical (e.g., the choice of optimizer in deep learning or the choice of heuristic or search operator in classical planning and meta-heuristics).

1.1 Algorithm Configuration

It is widely recognized that appropriate parameter settings are often instrumental for AI algorithms to reach a desired performance level (Hutter et al., 2010; Probst et al., 2019). In this paper, we will use the term *algorithm configuration* (AC) to refer to the process of determining a policy for setting algorithm parameters as to maximize performance across a problem instance distribution. AC has been widely studied, both in general (Birattari et al., 2002; Hutter et al., 2009; Ansótegui et al., 2009; Hutter et al., 2011; López-Ibáñez et al., 2016), as well as in specific research communities (Lobo et al., 2007; Snoek et al., 2012; Feurer & Hutter, 2019).

In this work, we focus on a particular kind of AC that is both (i) automated and (ii) dynamic. This general framework was recently proposed in a conference paper by Biedenkapp et al. (2020), and in this article we provide the first comprehensive treatment of the topic. In the remainder of this subsection, we contrast the dynamic/static and automated/manual approaches and position automated dynamic AC as a natural next step.

Dynamic vs. Static AC: In *static* AC, parameter settings are *fixed prior to execution*, using the information available at that time, and remain invariant during execution. For example, in evolutionary optimization, the population size is commonly set statically, e.g., as a function of the input dimensionality. In contrast, in *dynamic* AC (DAC), parameter settings are *varied during execution* using information that becomes available at run time. For example, in machine learning, while static AC would choose a learning rate, possibly dependent on meta-data (e.g., size or modality of the dataset), DAC would propose a learning rate *schedule* that could additionally be a function of time, alignment of past gradients, training/validation losses, etc. While not all parameters can be varied dynamically, in practice many can, and it often makes sense to do so. As a general motivating use case, consider parameters that (indirectly) control the exploration/exploitation trade-off: Typically, it makes sense to explore more early on, and to exploit this knowledge in later stages. Even if the optimal configuration happens to be static, predicting it *upfront* may be very hard, yet the best static configuration may quickly become apparent *while solving* the problem. For instance, if our learning rate is too high, training loss may diverge (Bengio, 2012). DAC has been an active research area that has produced various highly practical algorithms leveraging dynamic parameter adaptation mechanisms to empirically outperform their static counter-parts, e.g., Reactive Tabu Search (Battiti & Tecchiolli, 1994), CMA-ES (Hansen et al., 2003), and Adam (Kingma & Ba, 2015). Beyond these empirical successes, the potential of DAC has also been shown theoretically (Moulines & Bach, 2011; Senior et al., 2013; van Rijn et al., 2018; Doerr & Doerr, 2020; Speck et al., 2021).

Automated vs. Manual AC: The difference between manual and automated AC is *who performs AC*: A human or a machine. Over the last two decades, a variety of general-purpose automated algorithm configurators have been proposed that effectively relieve users from the tedious and time-consuming task of optimizing parameter settings manually (Hutter et al., 2009; Ansótegui et al., 2009; Kadioglu et al., 2010; Xu et al., 2010; Hutter et al., 2011; Seipp et al., 2015; López-Ibáñez et al., 2016; Falkner et al., 2018; Pushak & Hoos, 2020). However, there is still a lot of untapped potential, as all of these tools perform static AC. In contrast, *dynamic* AC is mostly done manually. Clearly, the human does not directly adjust the parameters during execution; rather, the mechanisms doing this auto-

matically, e.g., learning rate schedules, are products of human engineering. In this work, we will consider deriving such dynamic configuration policies in an automated and data-driven fashion.

1.2 Summary of Contributions

In this article, we provide the first comprehensive account of automated DAC. It subsumes and extends four prior conference papers, in which we

1. established DAC as a new meta-algorithmic framework and proposed solving it using contextual reinforcement learning (Biedenkapp et al., 2020);
2. applied DAC to evolutionary optimization, tackling the problem of step-size adaptation in CMA-ES (Hansen et al., 2003), and showed that existing manually-designed heuristics can be used to guide learning of DAC policies (Shala et al., 2020);
3. applied DAC to AI planning, tackling the problem of heuristic selection in FastDownward (Helmert, 2006), and showed how DAC subsumes static algorithm configuration and can improve upon the best possible algorithm selector (Speck et al., 2021); and
4. presented DACBench, the first benchmark library for DAC, facilitating reproducible results through a unified interface (Eimer et al., 2021b).

Here, we go well beyond this previous work, by

i more thoroughly discussing and classifying related work in different areas (Section 2), placing recent work on automated DAC in its scientific and historical context;

ii establishing a formal problem formulation (Section 3), offering a novel theoretical perspective on DAC and its relation to existing computational problems;

iii discussing possible methods for solving DAC problems (Section 4), beyond reinforcement learning, and classifying previous work according to their methodology;

iv extending and using DACBench (Section 5) to perform empirical case studies that

- demonstrate recent successes of automated DAC
- provide empirical validation for the benchmark library, and
- show that DAC presents a practical alternative to static AC, in various areas of AI: evolutionary optimization (Section 6.1), AI planning (Section 6.2), and machine learning (Section 6.3); and

v discussing current limitations of DAC (Section 7).

As such, we provide the first comprehensive overview of automated DAC, a standard reference and a solid foundation for future research in this area.

2. Related Work

Automated DAC is a new and exciting research area. However, it did not arise out of thin air, rather it closely relates to, builds on, and tries to consolidate past research efforts. In this section, we place recent work on automated DAC in its scientific and historical context. We start by introducing the terminology we use (Section 2.1). Then, we situate DAC in the broader context of AI (Section 2.2). Finally, we discuss some specific prior-art, covering historical work as well as the most recent developments (Section 2.3).

2.1 Terminology

Algorithm parameters are omnipresent in computer science. Unsurprisingly, no single set of terms has been consistently used when discussing the problem of how to best set them. In this section, we briefly clarify some of the terms we use, relating them to known alternatives.

We use the term *algorithm configuration* (AC) to refer to the process of determining a policy for setting an algorithm's parameters as to maximize performance (or equivalently, minimize cost) across an input distribution. In the classical AC literature (Birattari et al., 2002; Hutter et al., 2009; Ansótegui et al., 2009), this process results in a single parameter setting (i.e., a complete assignment of values to parameters) and is called a *configuration*. Later work generalized AC to produce configurations that are a function of the context in which they are used, e.g., the problem instance at hand (Kadioglu et al., 2010; Xu et al., 2010), and most recently the dynamic execution state (Biedenkapp et al., 2020). We will use the term *configuration policy* to refer to the result of AC in general. To disambiguate the aforementioned AC variants, we add the prefixes *per-distribution* (or also *classical*), *per-instance* and *dynamic*, respectively. Finally, while AC terminology was introduced in the context of attempts to automate this process, the term itself does not imply automation, i.e., we add prefixes *automated* and *manual* to specify whether configuration policies are determined automatically or through a manual engineering process, respectively.

In this work, we follow a meta-algorithmic approach to automating AC: We will treat AC as a computational problem to be solved by executing an algorithm. Hence, we have problem instances and algorithms at two different levels and will use the prefixes *(D)AC* and *target* to disambiguate these: For example, research on automated DAC aims to find a DAC algorithm for tackling the general DAC problem. In a given DAC problem instance, we aim to find a policy for configuring the parameters of a given target algorithm as to optimize its performance across a distribution of target problem instances. We also use *DAC method* and *DAC scenario* as a synonym for DAC algorithm and DAC problem instance, respectively.

In machine learning, the problem of setting the hyperparameters of the learning pipeline is known as hyperparameter optimization (HPO, Feurer & Hutter, 2019). We consider the more general problem of setting the parameters of *any* target algorithm and therefore adopt a more general terminology (Eggensperger et al., 2018). In meta-learning terms, *AC problem solving* corresponds to the *outer-loop* and *target problem solving* to the *inner-loop*.

In heuristic optimization, the terms *parameter tuning* and *parameter control* are commonly used to refer to static and dynamic algorithm configuration, respectively (Lobo et al., 2007). Also, the terms *online* (during use) and *offline* (before use) are sometimes used as synonyms for *dynamic* and *static*, respectively. In this work, we refrain from doing so, reserving these terms to refer to *when (D)AC takes place* (see Figure 1). In the offline setting, AC takes place in a dedicated *configuration phase* (similar to training in machine learning) where we determine which configuration to use later to solve the problems of actual interest to the user (i.e., at *use time*). In the online setting, AC happens at use time (Fitzgerald, 2021). In that sense, offline and dynamic are not mutually exclusive. In fact, most prior-art does DAC offline, determining a dynamic policy offline by using a training set, and at use time simply executing that dynamic policy on new problem instances.

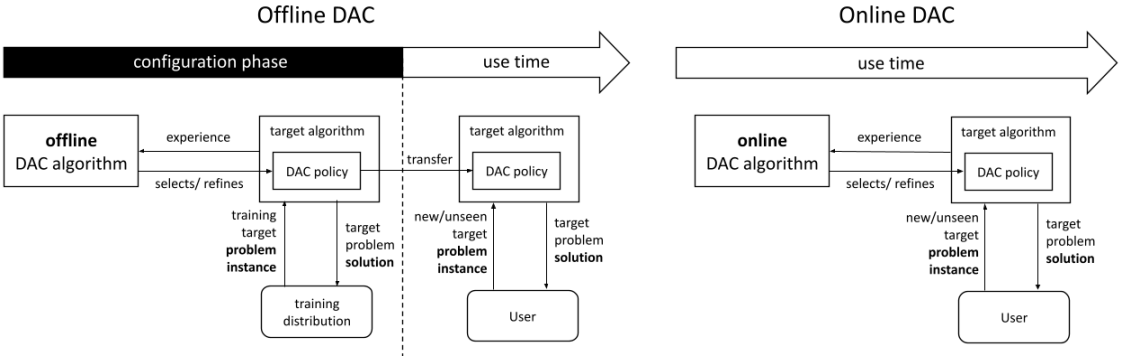


Figure 1: Offline vs. online learning of DAC policies.

2.2 Related Research Areas

While automating DAC is a relatively understudied problem, much research has been performed studying related problems. In what follows, we briefly characterize this work and how it relates to automating DAC. See Appendix A for a more formal treatment of this topic, where we provide problem definitions, possible reductions, and proof their correctness.

2.2.1 Automated Design of Algorithms / Components

The idea of letting computers, rather than humans, design algorithms has been studied in many different communities, using a variety of different methods. Some well-known, historical examples are *program synthesis*, using logical inference (Manna & Waldinger, 1980), and *genetic programming*, using evolutionary algorithms (Koza, 1992). Recent advances in machine learning have prompted a surge in approaches *learning* algorithms, e.g., *Neural Turing machines* (Graves et al., 2014), *learning-to-learn* (L2L , Andrychowicz et al., 2016; Lv et al., 2017; Bello et al., 2017; Metz et al., 2020), and *learning-to-optimize* (L2O, Li & Malik, 2017; Kool et al., 2018; Chen et al., 2021).

Generally speaking, algorithm parameters can be seen as "algorithmic design choices" that are left open at design time. In that sense, automated configuration is naturally viewed as a way of automating *part of* the algorithm design process. This approach has been referred to as "programming by optimization" (PbO, Hoos, 2012). While previous PbO applications used static AC approaches, the original PbO philosophy envisioned the possibility of varying design decisions at runtime, something naturally achieved by DAC.

A key difference between PbO and the aforementioned design automation approaches is that in PbO algorithms are not designed "from scratch", instead only design choices that are "difficult" for the human designer are made automatically by the configurator. For instance, DAC aims to design learning rate schedules (e.g., Daniel et al., 2016), but not entire optimizers as in L2L/L2O. In summary, DAC can be viewed as automatically designing parameter controlling components, and "DAC powered PbO" as a general *semi-automated* algorithm design approach that enables the human designer to bias the design process by embedding prior knowledge (e.g., obtained through decades of algorithmic research), thereby reducing the computational requirements and improving generalization.

2.2.2 Meta-Algorithmic Frameworks

Algorithm Selection Problems can be solved using a variety of different algorithms. For example, if we want to sort a sequence of numbers, we could do so using insertion sort, merge sort, quick sort, etc. In *algorithm selection*, we determine a mapping from features of the problem instance (e.g., sequence length) to the algorithm best suited to solve it (e.g., that sorts the sequence fastest). While first formalized by Rice (1976), this computational problem only received wide-spread attention two decades later, when it was independently rediscovered by Fink (1998), and Leyton-Brown et al. (2003) proposed to solve it using machine learning methods. This approach has resulted in various successful applications, e.g., SATzilla (Xu et al., 2008), a portfolio solver selecting between state-of-the-art SAT solvers to win multiple (gold) medals at the 2007 and 2009 SAT competitions. We refer to Kotthoff (2014) and Kerschke et al. (2019) for surveys on this topic.

Algorithm Scheduling It is often difficult to efficiently predict which algorithm will perform best on a given problem instance. In many settings, poor choices may require orders of magnitude longer than optimal choices, and tend to dominate average performance. In *algorithm scheduling*, instead of selecting a single algorithm, we aim to find an optimal time allocation. Automated algorithm scheduling was first extensively studied in seminal work by Huberman et al. (1997) and Gomes and Selman (2001), and follow-up work, e.g., by Hoos et al. (2015), typically focuses on finding a fixed time allocation that works best on average across instances (i.e., per-distribution). These kind of algorithm schedules are also very popular in the AI planning community, e.g., in Fast Downward Stonesoup (Helmert et al., 2011). Scheduling has also been combined with algorithm selection to find instance-specific schedules (Kadioglu et al., 2011; Lindauer et al., 2016). Dynamic scheduling approaches allocate resources to the algorithms based on runtime information (e.g., Carchrae & Beck, 2004; Gagliolo & Schmidhuber, 2006; Nguyen et al., 2021). This allows them to exploit the fact that, while it may be difficult to predict which algorithm performs best *in advance*, their relative performance may become apparent early-on in their executions. DAC also takes advantage of this property. However, unlike DAC, dynamic scheduling is restricted to allocating resources to *independent* processes; i.e., in scheduling, no information is exchanged between algorithm runs, and resources allocated to all but the one producing the eventual solution are effectively wasted.

Algorithm Configuration While algorithm selection chooses between multiple target algorithms on a per-instance basis, classical per-distribution algorithm configuration (AC) is concerned with finding the parameter setting of a single algorithm that performs best across all given instances. As the space of possible configurations grows exponentially in terms of the number of parameters, research on AC has traditionally focused on (i) efficient search methods, e.g., local search (Hutter et al., 2009), genetic algorithms (Ansótegui et al., 2009) and Bayesian optimization (Hutter et al., 2011); and (ii) efficient evaluation of configurations, e.g., using racing (Birattari et al., 2002), adaptive capping (Hutter et al., 2009), structured procrastination (Kleinberg et al., 2017) and multi-fidelity optimization (Li et al., 2018). This line of work has resulted in a variety of automated tools known as *configurators* that for any given target algorithm quickly find a configuration that performs well *on average* across a set of target problem instances, e.g., ParamILS (Hutter et al., 2009), GGA (Ansótegui et al., 2009, 2015, 2021), SMAC (Hutter et al., 2011), iRace (López-

Ibáñez et al., 2016), and Golden Parameter Search (Pushak & Hoos, 2020); as well as various theoretical insights (Kleinberg et al., 2017; Weisz et al., 2019; Hall et al., 2019, 2020). Configuration has further been combined with algorithm selection (Kadioglu et al., 2010; Xu et al., 2010), and algorithm scheduling (Seipp et al., 2015). However, all of these consider determining a static configuration policy, and the pursuit of similar automated tools and theory for DAC is a natural extension of this line of work.

2.2.3 Adaptive Operator Selection and Parameter Control

Heuristic Approaches The potential of varying parameters during execution time is widely recognized and has been extensively studied in various areas of AI. For instance, in heuristic optimization, this problem has been studied in the context of parameter control for evolutionary algorithms (Aleti & Moser, 2016), reactive search (Battiti et al., 2008), and selection hyper-heuristics (Drake et al., 2020). In machine learning, one hyperparameter that is typically varied is the learning rate, e.g., using global learning rate schedules (Loshchilov & Hutter, 2017; Smith, 2017) or adaptive gradient methods (Kingma & Ba, 2015) adopting weight-specific step-sizes. These works typically consider the dynamic configuration policy as a given and present an empirical/theoretical analysis thereof. Furthermore, the policies themselves were designed by human experts. In contrast, automated DAC is concerned with finding such policies automatically in a data-driven fashion. That being said, prior-art automating DAC does exist and is discussed in Section 2.3. Before doing so, we will briefly discuss a broad class of methods that rely less on human expert knowledge, but that we nonetheless do not generally regard as automated DAC.

Online Learning Approaches Many parameter control mechanisms integrate complex feedback loops, learning and optimization mechanisms, creating the potential that the DAC policy is not entirely predetermined by the human, but is rather learned online, while solving the problem instance at hand. All depends on the relative contribution to performance due to (i) the exploration of the hand-crafted DAC algorithm, and (ii) the exploitation of the DAC policy it learns. In an offline setting, distinguishing between (i) and (ii) is easy, as (i) does not occur at test/use time. In online settings, both are intertwined by nature. Note that this does not rule out "online DAC", but rather necessitates dedicated analysis that learning indeed takes place. Furthermore, in Section 3.2, we will define DAC as the problem of finding dynamic configuration policies "that generalize across a distribution of target problem instances". Therefore, in our nomenclature, *in order to qualify as automated DAC, an approach must demonstrate the ability to successfully transfer experience across runs of the target algorithm on target problem instances drawn from the same distribution.* In machine learning terms, automated DAC does not only require learning, but also *meta-learning*. Please note that most previous online learning approaches to parameter control (e.g., Muller et al., 2002; Carchrae & Beck, 2004; Chen et al., 2005; Eiben et al., 2006; Prestwich, 2008; Wessing et al., 2011; Gaspero & Urli, 2012; Schaul et al., 2013; Karafotias et al., 2014; Baydin et al., 2018) trivially do not meet this criterion, as no information is transferred across runs. Note that massive parallel online HPO methods such as Population Based Training (PBT, Jaderberg et al., 2017) also fall into this category.

2.3 Prior-Art: Automated Dynamic Algorithm Configuration

The term *dynamic algorithm configuration* (DAC) was only recently introduced by Biedenkapp et al. (2020). However, various authors had previously (or, in a few cases, concurrently) investigated the possibility of automatically determining policies for varying the configuration of an algorithm on-the-fly. In what follows, we give a brief overview of literature on automated DAC ("avant-la-lettre").[1] Here, we discuss these by application domain, a methodological overview is presented in Section 4.

Pioneering work by Lagoudakis and Littman (2000, 2001) explored this setting in the context of recursive algorithm selection, observing that sub-problems are better solved using different algorithms (e.g., sorting sub-sequences using different sorting algorithms). While initial results were promising, their approach was limited to recursive target algorithms. Pettinger and Everson (2002) considered a more general setting, learning a policy jointly selecting mutation and crossover operators in a genetic algorithm, per generation, based on statistics of the current population.[2] Various other works have explored automating DAC in the context of genetic algorithms (Fialho et al., 2010; Sakurai et al., 2010; Andersson et al., 2016), evolutionary strategies (Sharma et al., 2019), and heuristic optimization in general (Battiti & Campigotto, 2012; López-Ibánez & Stützle, 2014; Ansótegui et al., 2017; Kadioglu et al., 2017; Sae-Dan et al., 2020). Similar investigations were also conducted in various other communities, e.g., machine learning (Daniel et al., 2016; Hansen, 2016; Fu, 2016; Xu et al., 2017, 2019; Almeida et al., 2021), AI planning (Gomoluch et al., 2019, 2020), exact search (Bhatia et al., 2021), and quadratic programming (Getzelman & Balaprakash, 2021; Ichnowski et al., 2021).

Biedenkapp et al. (2020) introduced DAC in an attempt to consolidate these isolated efforts and to raise the level of generality in pursuit of algorithms similar to those that exist for static AC. Direct follow-up work has provided additional evidence for the practicality of DAC by learning step-size adaptation in CMA-ES (Shala et al., 2020), and by learning to select heuristics in the FastDownward planner (Speck et al., 2021). These application domains, together with the learning rate control setting from (Daniel et al., 2016), have later been released as part of a benchmark suite, called DACbench (Eimer et al., 2021b), offering a unified interface that facilitates comparisons between different DAC methods across different DAC scenarios. In this article, we extend this initial discussion of Biedenkapp et al. (2020) and present a thorough empirical comparison of AC and DAC on these three different real-world DAC applications (Daniel et al., 2016; Shala et al., 2020; Speck et al., 2021) using the unified DACbench interface.

3. Problem Definition

In this section, we formalize the computational problem underlying DAC. Here, we first introduce formulations for static AC variants (Section 3.1), and then define the dynamic AC problem (Section 3.2).

1. We maintain a list of work on automated DAC here:
   `https://www.automl.org/automated-algorithm-design/dac/literature-overview/`
2. Notably, direct follow-up work by Chen et al. (2005), no longer transferred experience across runs and is therefore not considered prior-art automating DAC (see Section 2.2.3).

3.1 Static Algorithm Configuration

In algorithm configuration, we have some target algorithm $\mathcal{A}$ with parameters $p_1, p_2, \ldots, p_k$ that we would like to configure, i.e., assign a value in the domains $\Theta_1, \Theta_2, \ldots, \Theta_k$, respectively. Furthermore, we may wish to exclude certain invalid combinations, giving rise to the space of candidate configurations $\Theta \subseteq \Theta_1 \times \Theta_2 \times \cdots \times \Theta_k$, called the *configuration space* of $\mathcal{A}$. In classical per-distribution algorithm configuration, we aim to determine a single $\boldsymbol{\theta}^* \in \Theta$ that minimizes a given cost metric c in expectation across instances $i \in I$ of our target problem distribution $\mathcal{D}$. This problem can be formalized as follows:

Definition 1: Classical / Per-distribution Algorithm Configuration (AC)

Given $\langle \mathcal{A}, \Theta, \mathcal{D}, c \rangle$:

- A target algorithm $\mathcal{A}$ with configuration space Θ
- A distribution $\mathcal{D}$ over target problem instances with domain I
- A cost metric $c : \Theta \times I \to \mathbb{R}$ assessing the cost of using $\mathcal{A}$ with $\boldsymbol{\theta} \in \Theta$ on $i \in I$

Find a $\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta} \in \Theta} \mathbb{E}_{i \sim \mathcal{D}} [c(\boldsymbol{\theta}, i)]$.

In practice, $\mathcal{A}$, $\mathcal{D}$, and c are not given in closed form. Instead, c is typically a black-box procedure that executes $\mathcal{A}$ with configuration $\boldsymbol{\theta}$ on a problem instance i and quantifies cost as a function of the desirability of this execution, e.g., how long the execution took, and/or the quality of the solution it found. Note that $\mathcal{D}$ is our true target distribution, i.e., the likelihood $\mathcal{A}$ is presented with an instance i at use time. In the online setting, we are given a sequence of samples from the actual distribution in real time (Fitzgerald, 2021). In the offline setting, we typically do not have access to $i \sim \mathcal{D}$, and are given a set of instances I' sampled i.i.d. from some representative training distribution $\mathcal{D}' \approx \mathcal{D}$ instead.

Note that unless a single configuration is non-dominated on all instances, better performance may be achieved by making the choice of $\boldsymbol{\theta}$ dependent on the problem instance i at hand. This extension is known as:

Definition 2: Per-instance Algorithm Configuration (PIAC)

Given $\langle \mathcal{A}, \Theta, \mathcal{D}, \Psi, c \rangle$:

- A target algorithm $\mathcal{A}$ with configuration space Θ
- A distribution $\mathcal{D}$ over target problem instances with domain I
- A space of *per-instance configuration policies* $\psi \in \Psi$ with $\psi : I \to \Theta$ that choose a configuration $\boldsymbol{\theta} \in \Theta$ for each instance $i \in I$.
- A cost metric $c : \Psi \times I \to \mathbb{R}$ assessing the cost of using $\mathcal{A}$ with $\psi \in \Psi$ on $i \in I$

Find a $\psi^* \in \arg\min_{\psi \in \Psi} \mathbb{E}_{i \sim \mathcal{D}} [c(\psi, i)]$

Note that the definition above is highly general. For example, by specifying Ψ accordingly PIAC can put arbitrary constraints on the configuration policies of interest. As a conse-

quence, classical per-distribution AC can be seen as a special case of PIAC only considering constant ψ, i.e., $\Psi = \{\psi \mid \psi(i) = \psi(i'), \forall i, i' \in I\}$. More generally, configuration policies could be restricted to be a function of specific features of i, or to belong to a specific (e.g., linear) function class. Note that this definition is also strictly more general than *unconstrained* PIAC, which is itself a special case. Also worth noting is that the cost metric c in this definition can be an arbitrary function of ψ (and i). In particular, we do not constrain c to be a function of $\psi(i)$, but allow it to quantify non-functional aspects of ψ, e.g., its minimal description length or computational complexity. That being said, most practical PIAC approaches are limited to minimizing $\mathbb{E}_{i\sim\mathcal{D}}[c'(\psi(i), i)]$, given some $c' : \Theta \times I \to \mathbb{R}$.

3.2 Dynamic Algorithm Configuration

In dynamic AC, we aim to optimally vary $\boldsymbol{\theta} \in \Theta$ while executing $\mathcal{A}$. In order to formalize this problem, we need to define points of interaction where $\mathcal{A}$ can be reconfigured. To this end, we decompose the execution of $\mathcal{A}$ with dynamic configuration policy $\pi \in \Pi$ on problem instance $i \in I$ as shown in Algorithm 1. Here, we start executing an "`init`" sub-routine bringing $\mathcal{A}$ in some initial state $s \in \mathcal{S}$ only depending on i. Subsequently, we iteratively execute "`step`" to determine the next state $s' \in \mathcal{S}$ of $\mathcal{A}$ as a function the current state s, instance i, and configuration $\pi(s, i) \in \Theta$. This process continues until `is_final`(s, i) signalling termination and s is returned as solution. When such decomposition $\langle$`init`, `step`, `is_final`$\rangle$ is given, we will call $\mathcal{A}$ *step-wise reconfigurable* and define DAC as follows:

Definition 3: Dynamic Algorithm Configuration (DAC)

Given $\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c \rangle$:

- A *step-wise reconfigurable* target algorithm $\mathcal{A}$ with configuration space Θ.
- A distribution $\mathcal{D}$ over target problem instances with domain I
- A space of *dynamic configuration policies* $\pi \in \Pi$ with $\pi : \mathcal{S} \times I \to \Theta$ that choose a configuration $\boldsymbol{\theta} \in \Theta$ for each instance $i \in I$ and state $s \in \mathcal{S}$ of $\mathcal{A}$
- A cost metric $c : \Pi \times I \to \mathbb{R}$ assessing the cost of using $\pi \in \Pi$ on $i \in I$.

Find a $\pi^* \in \arg\min_{\pi\in\Pi} \mathbb{E}_{i\sim\mathcal{D}}[c(\pi, i)]$

Algorithm 1 Step-wise execution of a dynamically configured target algorithm $\mathcal{A}$

Input: Dynamic configuration policy $\pi \in \Pi$; target problem instance $i \in I$
Output: Solution for i found by $\mathcal{A}$ (following π)

```
1: procedure A(π, i)
2:     s ← init(i)                    ▷ Initial state by starting the execution of A on i
3:     while ¬ is_final(s, i) do
4:         θ ← π(s, i)                ▷ Reconfiguration point: Use π to choose next θ
5:         s ← step(s, i, θ)          ▷ Continue executing A using θ
6:     return s                       ▷ Execution terminated: Return solution
```

Here, we define DAC as a generalization of PIAC, considering configuration policies that do not only depend on i, but also the dynamically changing state $s \in \mathcal{S}$ of the target algorithm $\mathcal{A}$, i.e., $\Psi \subseteq \{\pi \,|\pi(i,s) = \pi(i,s'), \forall\, s, s' \in \mathcal{S}, \forall\, i \in I\}$. This dynamic state, by definition, provides all information required for continuing the execution of $\mathcal{A}$, however can additionally contain arbitrary features of the execution thus far. As in PIAC, c can be an arbitrary function of π (and i). However, often the total cost of executing $\mathcal{A}$ with π on i can be decomposed and attributed to the T individual execution steps. Formally: In DAC with *step-wise decomposable cost*, we are given functions $\langle \mathrm{c}_{\text{init}}, \mathrm{c}_{\text{step}} \rangle$, such that

$$c(\pi, i) = \mathrm{c}_{\text{init}}(i) + \sum_{t=0}^{T-1} \mathrm{c}_{\text{step}}(s_t, i, \pi(s_t, i))$$

$$\text{where} \qquad s_t = \begin{cases} \texttt{init}(i) & t = 0 \\ \texttt{step}(s_{t-1}, i, \pi(s_{t-1}, i)) & t > 0 \end{cases} \quad \wedge \quad \texttt{is\_final}(s_t, i) \Leftrightarrow t = T.$$

Note that c_{init} and c_{step} only depend on i and $\pi(s_t, i)$, i.e., cannot measure non-functional aspects of π.

4. Solution Methods

In this section, we discuss methods for solving DAC. As discussed in Section 2.2.3, DAC has so far been primarily solved manually, i.e., dynamic configuration policies have been determined by humans and not in an automatic and data-driven way. In Section 2.3, we discussed previous work exploring *automated* DAC, and in what follows we will give an overview of the methods they used for doing so. Please note that no dedicated, general DAC solvers exist to date. Instead, prior-art can be viewed as solving DAC *by reduction* to some other well-studied computational problem.[3] Considering the fact that most of this work has been performed in isolation and tackles very different DAC scenarios, the high-level solution approaches followed are remarkably similar. In particular, we will roughly distinguish between two approaches: "DAC by reinforcement learning" (Section 4.1) and "DAC by optimization" (Section 4.2), and discuss their relative strengths and weaknesses (Section 4.3).

4.1 DAC by Reinforcement Learning

In reinforcement learning (RL, Sutton & Barto, 2018), an agent learns to optimize an unknown reward signal by means of interaction with an unknown environment. The RL agent takes actions $a \in A$, observes a transition T from the current state $s \in S$ of the environment to $T(s, a) \in S$, receives a reward $R(s, a) \in \mathbb{R}$, and learns for any state the action maximizing its expected future reward. Formally, the RL agent solves a Markov decision problem $\langle S, A, T, R \rangle$ (MDP, Definition 10 in Appendix A.3.5). Here, the transition T and reward R are given in the form of a black box method. Also, the state space S is typically not given explicitly; instead, we are given a procedure for generating initial states and can generate further states using T.

3. In Appendix A, we define these related computational problems and discuss reductions more formally.

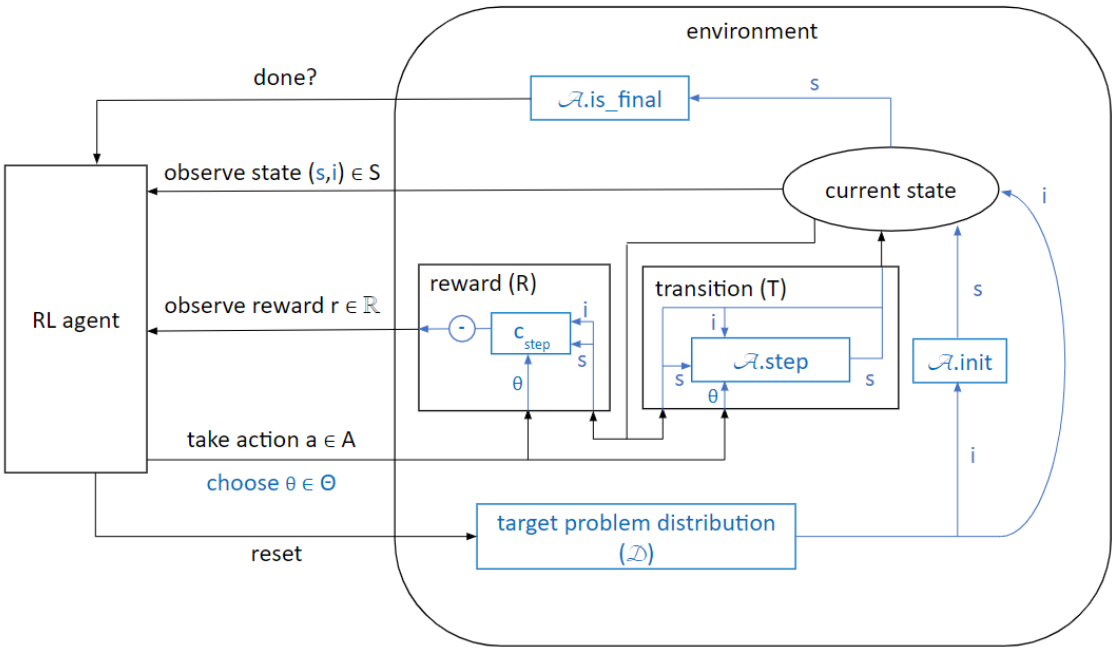


Figure 2: Illustration of DAC by Reinforcement Learning (DAC components in blue)

The RL problem described above is closely related to DAC, and prior art has commonly solved DAC using reinforcement learning methods. In DAC by RL, the environment consists of the target algorithm $\mathcal{A}$ solving some target problem instance $i \in I$. The state of this environment is $s = (s', i) \in S$ with $s' \in \mathcal{S}$ the state of the algorithm, and initial states $(\texttt{init}(i), i)$ with $i \sim \mathcal{D}$. At every reconfiguration point, the RL agent interacts with this algorithm choosing a configuration $\theta \in \Theta$ as action. The transition dynamics of the environment are fully determined by step-wise algorithm execution, i.e., $T((s', i), \theta) = (\texttt{step}(s', i, \theta), i)$, and the reward is $R((s', i), \theta) = -\,\mathrm{c}_{\text{step}}(s', i, \theta)$. See Figure 2 for an illustration of this approach. The power of this reduction lies in the fact that the resulting MDP can be solved using the full gamut of existing reinforcement learning methods.

Traditional RL: Early *DAC by RL* work (e.g., Lagoudakis & Littman, 2000, 2001; Pettinger & Everson, 2002; Sakurai et al., 2010; Battiti & Campigotto, 2012) used traditional value-based RL methods that learn the optimal state-action value function $Q^*(s, a)$ and return the policy $\pi(s) \in \arg\max_{a \in A} Q^*(s, a)$. These methods work well when $S \times A$ is small enough to be represented explicitly by a table, but do not scale up. Note that both S and $A = \Theta$ are typically too large in DAC to be modelled in tables.

Modern RL: Over the last decade, a series of methodological advances have given rise to a new generation of RL methods that can tackle complex real-world problems (Mnih et al., 2015; Silver et al., 2016; Barozet et al., 2020; Lee et al., 2020), and that have also been successfully applied to DAC. In particular, modern RL methods based on deep neural networks can effectively learn useful representations that allow them to handle complex state and action spaces, using, e.g., (double) deep Q-learning (DDQN, Hansen, 2016; Sharma

et al., 2019; Speck et al., 2021; Bhatia et al., 2021), modern actor critic (Andersson et al., 2016; Xu et al., 2017; Ichnowski et al., 2021), and policy gradient methods (Daniel et al., 2016; Xu et al., 2019; Gomoluch et al., 2019; Shala et al., 2020; Getzelman & Balaprakash, 2021; Almeida et al., 2021).

Contextual RL: It is worth noting that standard RL methods are not *instance-aware* and will generally not choose their initial state (see Figure 2, where i is hidden inside the environment). This is one of the reasons Biedenkapp et al. (2020) proposed to model DAC as a *contextual* MDP (cMDP, Hallak et al., 2015), which consists of a collection of MDPs, one for each instance i (see Definition 11 in Appendix A.3.5). Each MDP $\mathcal{M}(i)$ shares a common action space S and state space A as in traditional RL, but possesses an instance-specific transition function T_i and reward function R_i. This more general formulation allows DAC practitioners to explicitly model variation between instances: Variations in transition dynamics model the differences in target algorithm behaviour between instances (i.e., how the target algorithm progresses in solving an instance) while different reward functions reflect the instance-specific objectives. Although a single MDP can capture these dependencies implicitly, the explicit model allows the contextual RL agent to directly exploit this knowledge. For example, instances may vary in difficulty. A contextual RL agent, being aware of different instances and their characteristics, can more easily learn this, allowing the agent to more accurately assign credit for high/low rewards to (i) following a good/poor policy or (ii) solving easy/hard instances. Furthermore, the agent can choose which MDP $\mathcal{M}(i)$ it interacts with, e.g., to gather more experience on harder instances (Klink et al., 2020; Eimer et al., 2021a).

4.2 DAC by Optimization

Not all prior art automating DAC has done so using reinforcement learning. Instead, some previous works can be viewed as reformulating DAC as a (non-sequential) optimization problem: Given a search space Π and an objective function $f(\pi) = \mathbb{E}_{i\sim\mathcal{D}}\left[c(\pi, i)\right]$, find $\pi^* \in \arg\min_{\pi\in\Pi} f(\pi)$. This approach is illustrated in Figure 3. Optimization covers a wide variety of different methods. In what follows, we give an overview of those used in prior art for "DAC by optimization", and distinguish between different variants of optimization depending on (i) search space representation, and (ii) what information about f is used.

Noisy Black Box Optimization: In black box optimization (BBO), the only interaction between f and the optimizer is through an evaluation procedure e that returns $f(\pi)$ for any given $\pi \in \Pi$. A wide variety of black box optimizers exist, specialized for particular kinds of representations. In the reduction, dynamic configuration policies can be represented in a variety of different ways. For example, prior-art (Gomoluch et al., 2020) represents policies as real-valued vectors that correspond to the weights of a neural network policy, and optimizes these using evolution strategies. It is worth noting that a similar approach is currently state-of-the-art in learning-to-learn (Metz et al., 2020) (see Section 2.2.1). However, one could go further and also vary the architecture and optimize directly in the space of neural networks, e.g., using methods from neuroevolution (Stanley & Miikkulainen, 2002; Stanley et al., 2021). Alternatively, one could follow a symbolic approach, representing policies as programs and use genetic programming (Koza, 1992). Remark that this freedom comes

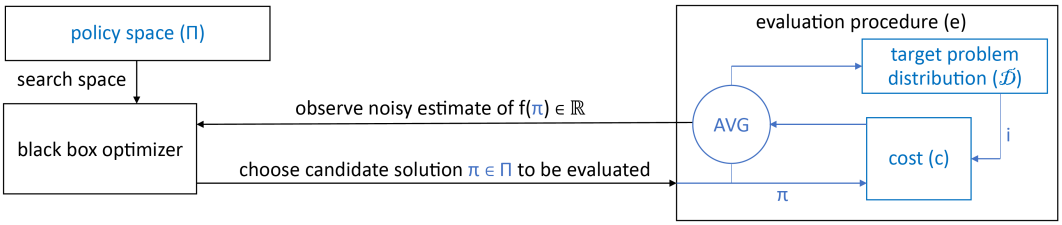


Figure 3: Illustration of DAC by Optimization (DAC components in blue)

with responsibility, i.e., making an appropriate choice of representation may be crucial to achieve satisfactory performance. Next to representation, another difficult choice in this reduction is the evaluation procedure. Since $\mathcal{D}$ is unknown, e cannot evaluate f exactly in general. Instead, we typically evaluate the cost on some finite sample of target problem instances $I' \subseteq I$ with $\forall i \in I' : i \sim \mathcal{D}$, and $e(\pi) = \frac{1}{|I'|} \sum_{i \in I'} c(\pi, i)$. However, the choice of $|I'|$ still poses a trade-off between accuracy and cost of evaluation to the DAC by BBO practitioner.

Static Algorithm Configuration: We can also solve DAC using classical static algorithm configurators (e.g., SMAC and irace). Assuming we choose a parametric representation Λ for the policy space, i.e., $\Pi = \{\pi_{\boldsymbol{\lambda}} \,|\, \boldsymbol{\lambda} \in \Lambda\}$, the DAC problem can be reformulated as classical AC, where we configure the parameters $\boldsymbol{\lambda}$ of the dynamic configuration policy $\pi_{\boldsymbol{\lambda}}$, instead of configuring the parameters $\boldsymbol{\theta}$ of the target algorithm.[4] While solving DAC using static AC may at first sight seem contradictory, this reduction gives rise to a highly practical solution approach that has been explored extensively in prior art (Fialho et al., 2010; López-Ibánez & Stützle, 2014; Andersson et al., 2016; Ansótegui et al., 2017; Kadioglu et al., 2017; Sae-Dan et al., 2020). An important benefit specific to this approach is that algorithm configurators are *instance-aware* and therefore automate the trade-off between the accuracy/cost of evaluation (using so-called *racing* mechanisms), and can even vary $I' \subset I$ dynamically to focus evaluation on those instances providing the most useful information.

Gradient-based Optimization In AC, we typically use gradient-free optimization. The motivation is that we cannot generally compute analytical gradients. While this is true *in general*, we would like to argue that the specific cases where we can actually compute them are more prominent than one might expect. Assuming a step-wise decomposable cost, we can compute the derivative $\nabla_{\boldsymbol{\lambda}} c_i = \frac{\partial c(\pi_{\boldsymbol{\lambda}}, i)}{\partial \boldsymbol{\lambda}}$ from the derivatives of the step-wise cost, the step, and the policy, using the chain rule (see Appendix A.3.6). When $\mathrm{c}_{\mathrm{step}}$, `step`, and π can be implemented using the operations in an automated differentiation framework (e.g., autograd in Pytorch, Paszke et al., 2017), these gradients can be calculated efficiently, reliably, without requiring any additional mathematical knowledge from the DAC practitioner. In fact, in the machine learning community, in particular meta-learning, differentiating through the entire learning process is almost standard practice (Maclaurin et al., 2015; Andrychowicz et al., 2016; Finn et al., 2017). The potential benefit of this extra piece of information is not to be underestimated. DAC policies may have many hyperparameters, e.g., a neural network with thousands of weights. Gradient-based optimization is an effi-

4. We proof the correctness of this reduction in Appendix A.3.1

cient way to navigate extremely high-dimensional spaces, as is evidenced by deep neural networks with millions of parameters being trained almost exclusively using simple first order optimization methods. That being said, gradients for DAC are no silver bullet. Computing them, while possible, may require too many computational resources. Furthermore, gradients only provide local information, i.e., an infinitesimal change to every parameter that is guaranteed to reduce cost. When f is particularly rugged, gradients may not provide information about the effect of any reasonably sized change. This phenomena has, in fact, been observed in the context of learning-to-learn (Metz et al., 2019).

4.3 Reinforcement Learning vs. Optimization?

Now, we discuss the relative strengths and weaknesses, and argue for the potential of combining both approaches.

Why DAC by RL? The sequential nature of the problem is arguably the key feature that sets DAC apart from static AC: In static AC, we only have to select a single configuration, while in DAC we must select a sequence of such configurations. RL provides a very general framework for tackling sequential decision problems and was presented as the method of choice for DAC by Biedenkapp et al. (2020). DAC by optimization approaches reduce DAC to a non-sequential optimization problem. In doing so, valuable information about the problem is lost that may otherwise be used to solve it more efficiently (Adriaensen & Nowé, 2016). While executing a target algorithm, an RL agent observes at *every step* what configurations were used, the (immediate) costs this incurred, and how this affected the dynamic state of the algorithm. In contrast, the same evaluation provides a black box optimizer with a single value (i.e., the sum of costs incurred), at the end of the run. This inherent relative sample-inefficiency of black box optimization is particularly problematic when target algorithm execution is costly, e.g., takes multiple hours.

Why DAC by Optimization? Previous work has shown that optimization can be a practical alternative to RL in simulated environments (Mannor et al., 2003; Szita & Lörincz, 2006; Salimans et al., 2017; Chrabaszcz et al., 2018; Majid, 2021). While RL aims to exploit sequential information, contemporary RL methods do not always do so successfully. Also, in some scenarios, this information may not add much value, or may even be deceptive (e.g., delayed rewards). Finally, these mechanisms add considerable computational overhead, and complicate implementation. In contrast, optimization methods tend to be simpler, have fewer failure modes, and their often parallel nature makes them well-suited for modern high-performance computing infrastructure. Adding to these limitations of RL methods are limitations of the reduction. While DAC is generally reducible to a (noisy) black box optimization problem, the previously discussed reduction to an MDP implicitely assumes (i) the cost function c to be step-wise decomposable an (ii) the space of policies Π to be unconstrained. As a consequence, it cannot be used when optimizing non-functional aspects of the policy (e.g., resources it requires to make decisions) or to impose arbitrary hard constraints on Π (e.g., which of these N policies is best?).

Beyond RL *or* Optimization: Our discussion thus far focused on contrasting both approaches. In what remains, we look at their relation, and argue for the potential of combining them. First, our "sequential vs. non-sequential" discussion can be extended to

"a method's ability to exploit a certain characteristic of DAC", or not. A good example of a cross-cutting characteristic is *instance-awareness*, both contextual RL and static AC can be viewed as instance-aware extensions of RL and black box optimization, respectively. Second, the pitfalls of RL also apply to approaches exploiting other characteristics. For example, gradients in optimization can be similarly deceptive (e.g, exploding/vanishing gradient problem) as immediate rewards. Therefore, while artificially hiding information is useless, blindly relying on it introduces failure modes, and general DAC methods should be carefully designed to only rely on information that is available and useful for the scenario at hand. In the context of "sequential vs. non-sequential", this suggests the importance of combining reinforcement learning *and* optimization. Further underpinning this conjecture, is the observation that state-of-the-art *static* AC methods combine optimization with machine learning, and reinforcement learning is essentially a dynamic extension of the latter.

5. Benchmark Library

In this section, we present DACBench (Eimer et al., 2021b), a novel benchmark library for DAC that we will be using in our experiments in Section 6. We have seen related fields like hyperparameter optimization, static algorithm configuration and algorithm selection profit greatly from focusing on shared benchmark problems (Eggensperger et al., 2013; Hutter et al., 2014; Bischl et al., 2016). In these meta-algorithmic domains, standardizing the target algorithm setup did not only increase the accessibility of the field by reducing some of the specialized knowledge required to get started in the field, it also made comparisons between different methods more reliable and reproducible. DACBench provides such a standard for DAC. In what follows, we give a brief overview of the interface it provides, the benchmarks it implements, and prior empirical validation it has undergone. We also discuss novel developments and highlight extensions that were motivated by and/or made specifically in the context of this work.[5]

Interface: DACBench builds upon a common RL interface, OpenAI's gym (Brockman et al., 2016), as it provides a flexible template for step-wise interaction with the target algorithm. The target algorithm `init` is handled in the `gym.Env.reset` method, with each step-wise interaction handled by the `gym.Env.step` method. DACBench extends the `gym.Env.reset` method to provide the ability to select the problem instance i to be solved. Listing 1 shows how DAC components are mapped onto the gym interface in the benchmarks. These essentially implement the DAC by contextual RL reduction, discussed in Section 4.1. The result is a simple-to-use interface, allowing DAC researchers to work across application domains, without requiring domain expertise, and providing an easy-to-use template for applying DAC to new domains. While the interface is modelled after the RL formulation of DAC, it can be used with a variety of approaches described in Section 4.2. That being said, the original DACbench interface strongly focused on conventional RL. In the scope of this work, we have extended the interface from the first release of DACBench. In accordance with our proposed definition of DAC, we have taken a broader perspective beyond standard RL, and made various interface changes to provide better support for alternative approaches. For example, users can now specify rich structured configuration

5. A new version of `https://github.com/automl/DACBench` (v. 0.1) was released alongside this article.

spaces as opposed to the simplistic action spaces supported by conventional RL methods. Directly controlling instance progression is easier now as well, providing a better base for developing instance-aware solution methods for DAC.

```
class DACEnv(gym.Env):

    def __init__(self, 𝒜, Θ, 𝒟, Π_ϕ, c_step):
        self.𝒜, self.𝒟, self.c_step, self.ϕ = 𝒜, 𝒟, c_step, Π_ϕ.ϕ
        self.action_space = Θ

    def reset(self, i=sample(self.𝒟)):
        s = self.𝒜.init(i)
        self.state = (s, i)
        return self.ϕ(s, i)

    def step(self, θ):
        s, i = self.state
        s = self.𝒜.step(s, i, θ)
        r = self.c_step(s, i, θ)
        done = self.𝒜.is_final(s, i)
        self.state = (s, i)
        return self.ϕ(s, i), r, done, None
```

Listing 1: A generic python implementation of a gym environment using components of a DAC scenario (in blue) with decomposable cost and an input-constrained policy space Π_ϕ. Practical DACBench benchmarks implement a similar mapping, but DAC components are typically not strictly separated, e.g., $\mathcal{A}.\texttt{step}$ and c_{step} would typically be calculated jointly. Note that the `gym.Env.step` method, despite its name, does far more than merely computing $\mathcal{A}.\texttt{step}$: It implements the transition dynamics (T) and reward signal (R). Furthermore, unlike $\mathcal{A}.\texttt{step}$, it is stateful, does not take the state (s, i) as input, and does not necessarily return the new state. Instead, it more generally returns what is called an observation $\phi(s, i)$ which may *abstract* arbitrary aspects of the internal state, i.e., DACBench technically reduces DAC to a contextual *partially observable* MDP (cPOMDP). Note that the learned policy in POMDPs is a function of the *observable* state, and hence ϕ can be viewed as modeling a policy space constraint of the form $\Pi_\phi = \{\pi \mid \phi(s, i) = \phi(s', i') \implies \pi(s, i) = \pi(s', i')\}$.

Benchmarks: An overview of the benchmarks currently included in DACBench is given in Table 1. It includes several benchmarks that we have either added in the latest release or at least improved significantly. The original SGD-DL benchmark (see Section 6.3 for a thorough description) was extended to mimic the experimental setup from Daniel et al. (2016) as closely as possible. The CMA-ES benchmarks (CMAStepSize and ModCMA) are now based on IOHProfiler (Doerr et al., 2018) and thus provide a DAC interface for a well-known and important tool in the EA community.[6] TheoryBench is a completely new benchmark, published by Biedenkapp et al. (2022), where one is to dynamically configure the mutation rate of a (1+1) random local search algorithm for the LeadingOnes problem.

6. The original *pycma* version of CMAStepSize is still supported and used in Section 6.1.

This is a particularly interesting setting as the exact runtime distribution is very well understood in this setting (Doerr, 2019). In particular, it is possible to compute optimal dynamic configuration policies for various different problem sizes and configuration spaces. Finally, a continuous Sigmoid variation and SGD on polynomials provide additional artificial benchmarks for efficient evaluation of DAC algorithms.

Empirical Validation: DACBench is a very recent library. As a consequence, it has not yet been used in prior-art. Eimer et al. (2021b) focused on providing a unified interface to a variety of benchmarks and analyzed specific properties of these benchmarks based on the behavior of static policies and simple hand-crafted dynamic policies. Here, we describe the first applications of practical DAC methods to these benchmarks, and provide important empirical validation for DACBench.

Benchmark	Domain	Status	Description
Sigmoid	Toy	Extended	Control k parameters to trace a different sigmoids each (Biedenkapp et al., 2020).
Luby	Toy	Original	Select the correct next term in a shifted luby sequence (Biedenkapp et al., 2020).
CMAStepSize	EA	Extended	Control the step size in CMA-ES (Shala et al., 2020).
FastDownward	Planning	Original	Control heuristic selection in FastDownward (Speck et al., 2021).
SGD-DL	DL	Extended	Control the SGD for neural network training (Daniel et al., 2016).
TheoryBench	EA	New	Control the mutation rate of (1+1)RLS for LeadingOnes (Biedenkapp et al., 2022).
ModCMA	EA	New	Control design choices (e.g., base sampler used) of CMA-ES (Vermetten et al., 2019).
ToyGD	Toy	New	Control the learning rate of gradient descent on polynomial functions.

Table 1: DACBench Benchmarks. "Status" compares the current state of each benchmark to the benchmarks originally introduced by Eimer et al. (2021b).

6. Empirical Case Studies

In this section, we discuss in more detail three successful applications of automated DAC in different areas of AI: evolutionary optimization (Shala et al., 2020, Section 6.1), AI planning (Speck et al., 2021, Section 6.2), and machine learning (Daniel et al., 2016, Section 6.3). The primary purpose of this section is to complement the general, big picture discussions in previous sections with some concrete practical examples of automated DAC. Here, we cover our own work in this area (Shala et al., 2020; Speck et al., 2021), supplemented with a machine learning application (Daniel et al., 2016) for diversity. In these case studies, we also conducted additional experiments to answer the following research questions.

RQ1: Can we reproduce the main results of the original papers using DACBench? Since it is well known that RL results are hard to reproduce (Henderson et al., 2018), in order to provide a solid foundation for experimental work in the field we believe it to be important to repeat the original experiments, this time using the publicly available re-implementations provided by DACBench (i.e., the CMAStepSize, FastDownward, and SGD-DL benchmarks) and to compare the results obtained to those of the original papers. Beyond insights into the reproducibility of the prior work, this analysis provides empirical validation for DACBench: This is the first study investigating whether, and to what extent, the benchmarks in DACBench permit reproducing the original results. Further, it is worth noting that the work by Daniel et al. (2016) is closed source, and that this is the first reproduction of their experiments with open-source code.

RQ2: Does DAC outperform static AC in practice?
Theoretically, an optimal DAC policy will be at least as good as an optimal static AC policy. In practice, however, the superiority of DAC is not guaranteed, since practical DAC methods may not be capable of finding an optimal/better DAC policy and/or doing so may require more computational resources than available. To investigate this, for each scenario in our case studies, we compare the anytime performance of the DAC method used to that of static AC baselines: We run SMAC (as a classical AC method, Hutter et al., 2011; Lindauer et al., 2022) and Hydra[7] (as a PIAC method, Xu et al., 2010) on the same problem, and compare the performance of the best dynamic/static policies found at any time during the configuration process. We further include the theoretical upper bounds for classical AC (SBS = $\min_{\boldsymbol{\theta}\in\Theta} \frac{1}{|I'|}\sum_{i\in I'} c(\boldsymbol{\theta}, i)$) and PIAC (VBS = $\frac{1}{|I'|}\sum_{i\in I'} \min_{\boldsymbol{\theta}\in\Theta} c(\boldsymbol{\theta}, i)$) as reference, to distinguish practical from inherent limitations of static AC.[8]

In what follows, we discuss our three case studies, in each case presenting an introduction to the domain, the problem formulation as an instance of DAC, the solution method, the experimental setup, the results, and a discussion thereof.[9]

6.1 Step Size Adaptation in CMA-ES

The first problem we consider is to dynamically set the step-size parameter of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES, Hansen et al., 2003), an evolutionary algorithm for continuous black box optimization. Each generation g, CMA-ES evaluates the objective value f of λ individuals $x_1^{(g+1)}, ..., x_\lambda^{(g+1)}$ sampled from a non-stationary multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}^{(g)}, {\sigma^{(g)}}^2 \cdot C^{(g)})$. Then, based on the outcome of these evaluations, CMA-ES heuristically adapts the parameters $\boldsymbol{\mu}$, σ, C of the search distribution aiming to increase the likelihood of generating better individuals next generation. In particular, the step-size parameter σ controls the scale of the search distribution and CMA-ES by default adjusts it using Cumulative Step Length Adaptation (CSA, Hansen & Ostermeier, 1996). CSA is a hand-designed heuristic and thus implicitly makes assumptions about the properties of the tasks it is applied on. In Shala et al. (2020), we investigated the possibility of learning step-size adaptation in a data-driven fashion, optimized for the task distribution at hand, i.e. automated DAC.

Problem Formulation: Below, we briefly detail each of the DAC components:

$\mathcal{A}$, Θ: The target algorithm to configure in this scenario is CMA-ES. As in Shala et al. (2020), we use the *pycma* distribution of CMA-ES. Its interface allows for step-wise execution of CMA-ES. CMA-ES is initialized with a given initial mean $\boldsymbol{\mu}^{(0)}$ and step-size $\sigma^{(0)}$ ($C^{(0)} = \mathbb{1}$). Each generation g, we

7. Hydra combines SMAC with an algorithm selection method of choice. Since most of the considered benchmarks do not have instance features, we will assume an oracle selecting the best configuration in the portfolio. We treat the maximum size of the portfolio as a case study dependent hyperparameter and detail this choice in the respective experimental setups.
8. Note that the acronyms SBS (single best solver) and VBS (virtual best solver) stem from the algorithm selection literature. More details on how these theoretical bounds were determined can be found in the experimental setup of the respective case studies.
9. Code for reproducing these experiments is publicly available: `https://github.com/automl/2022_JAIR_DAC_experiments`

1. sample λ individuals $x_1^{(g+1)}, ..., x_\lambda^{(g+1)}$ from $\mathcal{N}(\boldsymbol{\mu}^{(g)}, {\sigma^{(g)}}^2 \cdot C^{(g)})$
2. evaluate the objective function values $f(x_1^{(g+1)}), ..., f(x_\lambda^{(g+1)})$ of these individuals
3. adapt the distribution parameters $\boldsymbol{\mu}^{(g+1)}$, $\sigma^{(g+1)}$, $C^{(g+1)}$ for the next generation.

In this final step, the mean $\boldsymbol{\mu}$ and covariance C are adapted as usual in CMA-ES, while the step-size σ is to be reconfigured dynamically in the range $\Theta = \mathbb{R}^+$.

$\mathcal{D}, I$**:** Instances correspond to tuples consisting of a black box function f and an initial search distribution. Here, the latter is isotropic and defined by an initial mean $m^{(0)}$ and step-size $\sigma^{(0)}$.

Π : The policies are constrained to be functions of a specific observable state composed of: (i) the current step-size value $\sigma^{(g)}$; (ii) the current cumulative path length $p_\sigma^{(g)}$ (Hansen & Ostermeier, 1996); (iii) the history of changes in objective value, i.e., the normalized differences between successive objective values, from h previous iterations; and (iv) the history of step-sizes from h previous iterations.

c : The cost metric used is "the likelihood of outperforming CSA". Assuming we perform two runs of CMA-ES, one using π, the other CSA, it measures how likely the latter is to obtain a better final solution than the former. We estimate this probability based on pairwise comparisons of $n = 25$ runs varying only the random seed of CMA-ES, i.e.,

$$c(\pi, i) = \frac{\sum_j^n \sum_k^n \mathbb{1}_{\pi_j < CSA_k}}{n^2}$$

where $\mathbb{1}_{\pi_j < CSA_k}$ is the function indicating that our policy resulted in a lower final objective value than the baseline using CSA, when comparing runs j and k. Note that a benefit of this cost metric is that it is easy to interpret, both conceptually and in terms of statistical significance. As explained in more detail in the original publication (Shala et al., 2020, Appendix C), it has a direct correspondence with the Wilcoxon rank sum statistic. For $n = 25$, estimates $c(\pi, i) \geq 0.64$ imply π significantly outperformed CSA (at 95% confidence, one-sided).

Solution Method: In Shala et al. (2020), we proposed to use existing hand-crafted heuristics to warm-start DAC. To this end, we adopted the methodology proposed by Li and Malik (2017) in the context of L2O and used guided policy search (GPS, Levine & Abbeel, 2014), a reinforcement learning method originating from the robotics community. In GPS, a teacher (typically a human) provides some initial sample trajectories that the RL agent first learns to imitate and then iteratively refines without further interaction with the teacher. To learn step-size adaptation policies, in Shala et al. (2020), we used CSA as a teacher and extended GPS with *persistent teaching*, meaning that at each iteration the GPS agent obtains a fixed fraction (the *sampling rate*, a hyperparameter) of its sample trajectories from the teacher, encouraging it to continually learn from CSA. Folowing Li and Malik, the area under the curve (AUC) was used as a reward signal for GPS, instead of negated cost. Here, the reward at step t is $-\min_{x \in X_t} f(x)$ where $X_t = \{x_i^{(g)} \mid g \leq t\}$ is the set of individuals evaluated up until step t. This reward signal, unlike negated cost, is dense and actively encourages learning policies with good anytime performance.

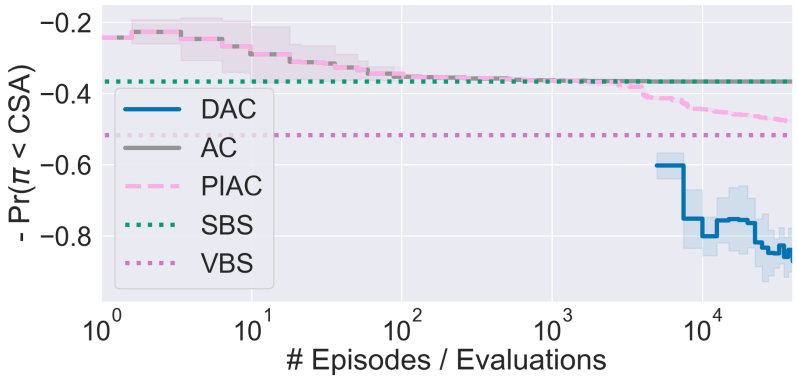


Figure 4: Incumbent performance of DAC (GPS), PIAC (Hydra), and classical AC (SMAC) when determining a step-size configuration policy for CMA-ES. Solid lines depict the mean of five independent configuration runs and the shaded area the standard deviation. SBS depicts the single best configuration and VBS the oracle configuration portfolio across all instances.

Experimental Setup: In our experiments, we used the DACBench implementation of the CMAStepSize benchmark. Replicating the original setup, we set population size $\lambda = 10$, history length $h = 40$, terminate CMA-ES after 50 generations, and model policies as fully connected feed-forward neural networks having two hidden layers with 50 hidden units each and ReLU activations. Note that in Shala et al. (2020), we considered a collection of different scenarios varying in target distribution: (i) single black box function, different initial search distributions; (ii) black box functions of the same type, but different dimensionalities and initial search distributions; and (iii) black box functions of different types and initial search distributions. In our case study here, we only reproduce and discuss the results for the third scenario, as it considers learning policies that generalize across different black box functions. Here, the training setup consists of 100 training instances: 10 different black box functions, with 10 different initial search distributions each. For testing, 12 other black box functions were used with a specific initial search distribution. In both cases, the functions used were taken from the BBOB-2009 competition (Hansen et al., 2009). We perform five independent GPS training runs using the original hyperparameters, each performing a total of 40000 CMA-ES runs and taking 8-10 CPU hours on our system. In our comparison of anytime performance to static AC, the same budget was used for classical AC (SMAC) and PIAC (Hydra). A maximum portfolio size of 10 was used for Hydra. To determine SBS and VBS, we discretized Θ (1000 values equidistant in [0.1, 2.0]) and evaluated $c(\boldsymbol{\theta}, i)$ for all (1000 $\times$ 100) combinations of $\boldsymbol{\theta} \in \Theta$ and $i \in I'$.

Results: Figure 4 compares the anytime training performance of DAC (GPS) to that of classical AC (SMAC) and PIAC (Hydra) when learning step-size adaptation. Classical AC and PIAC initially show similar anytime behavior, where the former reaches SBS performance after 1000 evaluations, the latter further improves, but does not reach VBS performance within the given budget of 40000 evaluations. In contrast, DAC (GPS) has

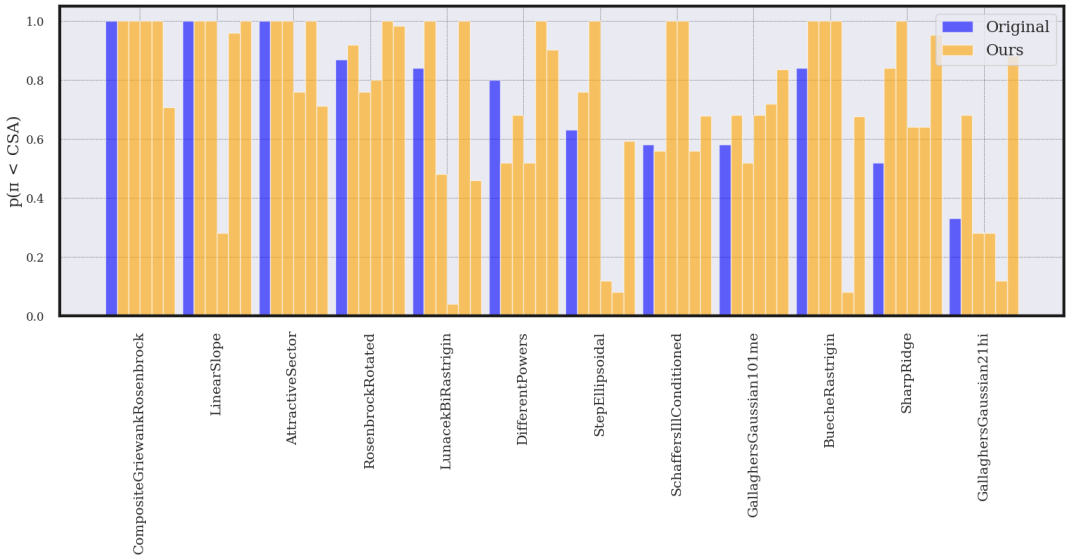


Figure 5: Likelihood of the policies learned by GPS (for five runs) outperforming CSA on 12 unseen test functions. The reported values from (Shala et al., 2020) are shown in blue, whereas the results for the five learned policies are shown in yellow, in a consistent order.

a minimum budget of 5000 evaluations, however, the initial incumbent immediately outperforms the VBS and further improves to eventually find a DAC policy that on average outperforms CSA on 87% of the runs on the training setting. Figure 5 shows the likelihood of the five learned policies outperforming CSA on the 12 unseen test functions. Here, for each of five individual seeds, we observe that the learned policies significantly ($p(\pi < \text{CSA}) \geq 0.64$, $\alpha = 0.05$) outperformed CSA on $10, 9, 10, 8, 8$ of the 12 unseen test functions, while being significantly outperformed on $0, 1, 0, 3, 3$, respectively. In comparison to the original, the learned policies performed similarly when averaging costs across all test functions/policies (0.74 vs 0.75 originally). However, it is worth noting that the average performance of the individual policies and the performance profile across the test functions varies more strongly.

Discussion: On a high level, we could reproduce our results from Shala et al. (2020), showing that the learned policies for step-size adaptation can outperform CSA even on functions not seen during training. Since the DACBench implementation, to the best of our knowledge, exactly replicates the original setup, we assume the observed differences to be a consequence of variability across training runs. This is supported by our observation that the five different runs of GPS (varying only in random seed) resulted in policies whose test cost ranged from -0.83 to -0.65 (vs. -0.75 originally). Our analysis of the anytime performance revealed another weakness of the approach: Its relatively high up front cost. It is worth noting that this cost includes the teacher runs (25×100 runs using CSA) we performed to warm-start GPS. Nonetheless, since GPS maintains an independent controller

per instance, its computational cost will generally scale linearly with the number of training instances. Further, it is difficult to predict in advance how many training instances and runs per training instance suffice. In comparison, the static approaches in our comparison follow a more incremental approach resulting in a better anytime performance. That being said, the best static policy did not significantly outperform CSA. As such, independent of the specific approaches, our results provide further evidence of the importance of dynamic step-size adaptation, showing that DAC policies (learned, but also CSA) are competitive with and/or outperform their static counterparts, even on relatively short CMA-ES runs.

6.2 Heuristic Selection in FastDownward

Heuristic search is one of the most widely used and successful approaches to AI planning. This type of search makes use of heuristics to estimate the distance to some desired goal state as a cheap proxy of having to directly evaluate the true distance. Over decades of research, many different heuristics have been developed for a variety of problem domains. No single heuristic works best on all problem instances (Wolpert & Macready, 1995). Thus, the AI planning community has made use of meta-algorithmic approaches such as algorithm selection, algorithm scheduling and algorithm configuration (Helmert et al., 2011; Seipp et al., 2014; Fawcett et al., 2014; Seipp et al., 2015; Sievers et al., 2019). However, one limiting factor of these approaches is that they do not take the internal dynamics of the planning system into account and only adapt to a set of problem instances (per-distribution) or individual problem instances (per-instance). It has been shown that using hand-crafted policies to switch between heuristics to adapt to changing conditions can greatly improve performance (Richter & Helmert, 2009; Röger & Helmert, 2010). Speck et al. (2021) proposed to use dynamic algorithm configuration (DAC) to automatically determine a policy that selects at each individual planning step which heuristic to follow, out of a set of heuristics sharing their progress. That work showed that DAC is in theory able to outperform prior meta-algorithmic approaches and empirically validated this by outperforming the theoretical best algorithm selector (a.k.a. virtual best solver) on multiple domains. Relatedly, Gomoluch et al. (2019, 2020) previously investigated automated DAC in the context of switching between different search strategies in AI planning. To provide an exemplary showcase of the potential of DAC in AI planning, we focus on the heuristic selection problem here.

Problem Formulation: Below, we briefly detail each of the DAC components:

$\mathcal{A}$, Θ: The target algorithm to configure in this scenario is the popular FastDownward Planner (Helmert, 2006). To make it step-wise executable, and to allow communication with a dynamic configuration policy, Speck et al. (2021) proposed to set up a socket communication such that DAC can change heuristics after each node expansion. The configuration space consists of four heuristics[10] (i.e., a single categorical parameter), commonly used in satisficing planning: (i) the FF heuristic h_{ff} (Hoffmann & Nebel, 2001), (ii) the causal graph heuristic h_{cg} (Helmert, 2004), (iii) the context-enhanced additive heuristic h_{cea} (Helmert & Geffner, 2008), and (iv) the additive heuristic

10. In an additional experiment, Speck et al. (2021) showed that even with an increased action space, including the landmark-count heuristic (Richter et al., 2008), DAC was still capable of learning better policies than the considered baselines. Here, we limit ourselves to the original configuration space which only includes four heuristics.

h_{add} (Bonet & Geffner, 2001). The planning system is terminated when a solution is found. Since some runs may fail to find a solution, Speck et al. (2021) also limited the maximal run length. During the configuration phase (training of the RL agent) an individual solution attempt can run for at most 7500 steps. During evaluation, this conservative step-limit of 7500 steps is removed and instead a maximum of five minutes running time is used.

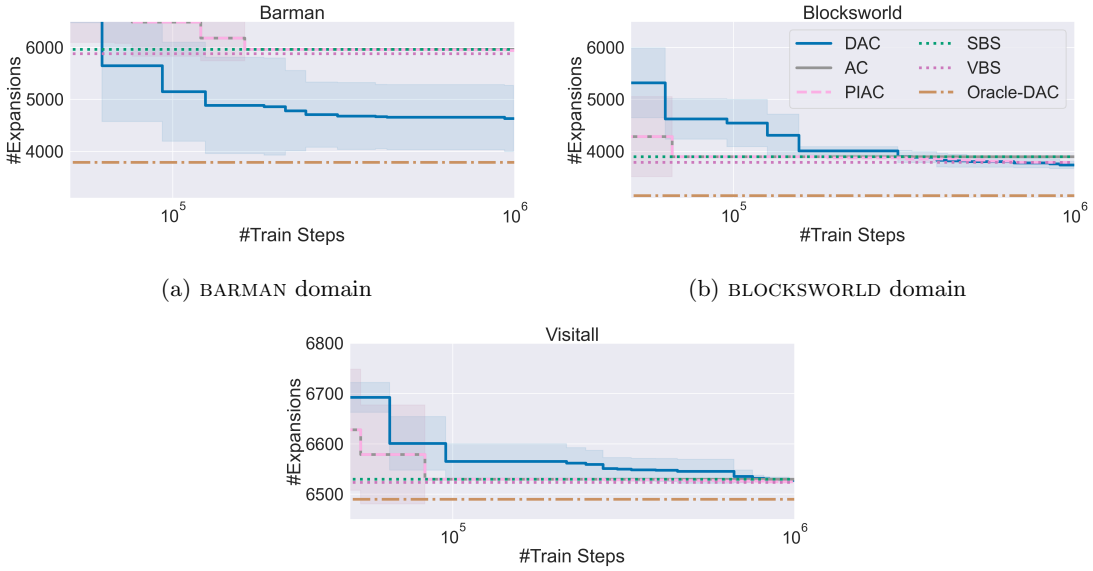
$\mathcal{D}, I$**:** The target problems consist of 100 training and 100 disjoint test problem instances taken from each of six different domains from the international planning competition (IPC). The instances, however, were not taken from a particular round of the IPC as some domains only contain few instances. Instead, Speck et al. (2021) used instance generators to generate instances that resemble those of the IPC tracks.

Π**:** The policies are constrained to be a function of a specified observable state. The observable state consists of simple statistics about the heuristics in the configuration space. Specifically, for every heuristic h, it contains the (i) maximum h value; (ii) minimum h value; (iii) average h value; (iv) variance of h over all possible next states; (v) number of possible next states as determined by h; and (vi) current expansion step t. In order to encode progress towards solving a problem instance, Speck et al. (2021) did not use these state features as is, but rather their change w.r.t. the previous step (i.e., the difference between consecutive observations).

c**:** The considered cost metric is the total number of node expansions, i.e., the number of planning steps until a solution is found. The decomposed cost metric is $+1$ for every step. Thus, configuration policies that minimize the average number of planning steps are preferential. Note that, given the termination criterion of $\mathcal{A}$, the maximal cost at configuration time is 7500, corresponding to not finding a solution in time. During evaluation, coverage is analyzed instead, i.e., the number of instances solved within the five minute budget.

Solution Method: The proposed solution approach by Speck et al. (2021) uses a small double deep Q-network (DDQN, van Hasselt et al., 2016) to learn a dynamic configuration policy via reinforcement learning. In our experiments, we use the original reinforcement learning code with the exact same hyperparameters as provided by the original authors. Since DACBench offers a standard RL interface (see Section 5), the original RL code could be reused without modification.

Experimental Setup: In our experiments, we make use of the implementation of the interface as provided via DACBench (FastDownward benchmark). Following Speck et al. (2021), we learn a separate policy for each domain, however, to reduce the computational cost, we limit ourselves to a representative set of three out of six domains. Following Speck et al. (2021), we perform five independent training runs for each domain. In each training run, an RL agent experiences 10^6 steps of the planning system, taking 8-12 hours on our system. Since $|\Theta| = 4$, SBS and VBS could be determined exactly for each domain by evaluating $c(\boldsymbol{\theta}, i)$ for all (4×100) combinations. For Hydra, we used a maximum portfolio size of three which is sufficient to cover the optimal portfolio.



(a) barman domain

(b) blocksworld domain

(c) visitall domain

Figure 6: Incumbent performance of DAC (DDQN), PIAC (Hydra), and classical AC (SMAC) when determining a heuristic selection policy for FastDownward on (a) the barman, (b) blocksworld, and (c) visitall domains. Solid lines depict the mean of five independent configuration runs and the shaded area the standard deviation. SBS depicts the single best configuration and VBS the oracle configuration portfolio across all instances. Oracle-DAC is the oracle portfolio of all dynamic policies evaluated by DAC (DDQN), providing a pessimistic performance estimate of optimal dynamic configuration policy.

Results: Figure 6 compares the anytime performance of DAC (DDQN) to that of classical AC (SMAC) and PIAC (Hydra) for all three domains. On the barman domain, DAC finds policies that on average clearly outperform the best static baseline in less than 10% of the total budget. On the blocksworld domain, DAC almost needs the full budget, but eventually finds policies that marginally outperform the VBS. The visitall domain is even slightly harder and DAC (DDQN) does not confidently find policies outperforming the static baselines within the limited budget. For lower budgets, classical AC and PIAC obtain clearly better policies on visitall / blocksworld, and PIAC eventually approaches VBS performance on both domains. Table 2 compares the coverage results for all learned policies on the test problem instances with a static baseline. Here, we find that our learned policies generalize well to the test scenarios and achieve similar coverage as reported in the original paper. In the barman domain, DAC policies dominate, and while we achieve a slightly lower coverage on this domain than originally, this can largely be attributed to an individual training run of ours performing worse than the others, with the individual coverages 85.00, 88.32, 67.00, 84.00, and 84.00. In the other two domains, we achieve slightly higher coverages than originally, and the DAC policies perform similarly well as the best static policies.

Algorithm	DAC POLICY						SINGLE HEURISTIC				AS ORACLE
Domain (# Inst.)	RL Run#1	Run#2	Run#3	Run#4	Run#5	RL†	h_{ff}	h_{cg}	h_{cea}	h_{add}	SINGLE h
BARMAN (100)	85.0	88.3	81.7 67.0	84.0	84.0	84.4	66.0	17.0	18.0	18.0	67.0
BLOCKSWORLD (100)	95.0	95.0	93.6 91.0	94.0	93.0	92.9	75.0	60.0	92.0	92.0	93.0
VISITALL (100)	58.1	56.1	58.6 57.8	60.0	61.0	56.9	37.0	60.0	60.0	60.0	60.0
SUM (300)			233.9			234.2	178.0	137.0	170.0	170.0	220.0

Table 2: Number of solved *unseen* test problem instances averaged over five independently repeated training runs. Column RL provides the results of our experiment, with the results of the individual runs given in a smaller font, whereas RL† contains the original coverage values as reported by Speck et al. (2021). All h_i columns contain the number of solved problem instances when only using the specific heuristic. AS ORACLE reports the coverage results of the theoretically best algorithm selector.

Discussion: Our results confirm the results of Speck et al. (2021) where the DAC policies (i) obtain slightly lower coverage than the single best heuristic in the VISITALL domain, (ii) outperform the single best heuristic and are close in performance to the theoretical best algorithm selector on the BLOCKSWORLD domain and (iii) provide the best coverage by far in the BARMAN domain. Most notably, on average, the learned DAC policies are capable of solving more problem instances than the theoretical best algorithm selector, which already provides a significant improvement over using the single best heuristic. Our analysis of the approach's anytime performance also revealed that when less time is available, static AC approaches, in particular PIAC (Hydra), outperform DAC (DDQN) on two of the three domains. However, on the remaining domain (BARMAN), superior dynamic policies are easily found. It is worth noting that on all three domains, oracle-DAC is clearly superior, suggesting the potential to further improve performance by using better DAC methods and/or more informative state features.

6.3 Learning Rate Control in Neural Network Training

Daniel et al. (2016) investigated meta-learning a controller for the learning rate hyperparameter η in Stochastic Gradient Descent (SGD) style neural network optimizers. SGD is the method of choice for optimizing the parameters $\boldsymbol{w}$ of deep neural networks, i.e., solve $\arg\min_{\boldsymbol{w}} L(\boldsymbol{w}, D)$, where L is some differentiable measure of loss on the training data D. In deep learning, it is common to have millions of parameters. To scale up to such extremely high-dimensional $\boldsymbol{w}$, SGD exploits the fact that $\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, D) = \frac{\partial L(\boldsymbol{w}, D)}{\partial \boldsymbol{w}}$ can be computed exactly, and updates $\boldsymbol{w}$ in the opposite direction of the gradient. As datasets in deep learning are huge, computing the "full batch" gradient is typically too expensive. Instead, SGD computes the gradient at every optimization step for a different randomly selected "mini-batch" $B \subset D$. While this gradient is an unbiased estimate of the actual gradient, i.e., $\mathbb{E}[\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, B)] = \nabla_{\boldsymbol{w}} L(w, D)$, variance can cause gradients to occasionally point in the wrong direction. Furthermore, gradients only provide local information and do not tell us

how far we can move without overshooting. Moreover, the optimal step sizes per dimension may vary strongly, a problem known as ill-conditioning. Over the last decade a variety of different variants of SGD, e.g., Momentum (Jacobs, 1988), RMSprop (Tieleman et al., 2012), and Adam (Kingma & Ba, 2015), have been proposed that aim to address these and other issues. However, despite their popularity, modern SGD variants are still sensitive to their hyperparameter settings. In particular, they still have a global/initial learning rate η, that uniformly scales the step taken in each dimension, and that must typically be optimized for the problem at hand (Bengio, 2012). When setting η too low, optimization is slow, while too high η might even lead to divergence. To the best of our knowledge, Daniel et al. (2016) was the first work that explored replacing η by a meta-learned controller, producing more robust SGD methods. Xu et al. (2019) followed up on this idea, and most recently Almeida et al. (2021) considered meta-learning the control of learning rate *and* various other hyperparameters (e.g., weight-decay and gradient clipping).

Problem Formulation: The meta-learning approach by Daniel et al. (2016) is readily seen as automated DAC. Below, we briefly detail each of the DAC components:

$\mathcal{A}$, Θ: Daniel et al. (2016) present a general method for dynamically configuring the learning rate $\eta_t \in \mathbb{R}^+$ at every optimization step of SGD. In their experiments, they do this for two SGD variants: RMSprop and Momentum.[11] Note that in the first optimization step, a fixed learning rate η_0 is used.

$\mathcal{D}, I$: Instances correspond to neural network optimization problems, and are represented by the quadruple $\langle D, L, k, \xi \rangle$, where

- D is the data we want to fit the neural network to. In their experiments, Daniel et al. (2016) consider image classification, using examples from the MNIST and CIFAR10 datasets.
- L is the differentiable loss function to be minimized. Daniel et al. used cross-entropy loss, i.e., the negative log-likelihood of the data D under the model with parameters w, where this model can be any parametric model. Daniel et al. (2016) used small convolutional neural networks (CNNs).
- k is the cutoff: SGD is terminated after k optimization steps.
- ξ is the seed of the pseudo-random number generator used for random neural network initialisation and mini-batch sampling.

Π : Daniel et al. (2016) considered dynamic configuration policies that are a log-linear function $\pi_{\boldsymbol{\lambda}}(\boldsymbol{\phi}) = \exp(\lambda_0 + \sum_{j=1}^{4} \lambda_j \phi_j)$ of four expert features $\boldsymbol{\phi}$ that in turn depend on the previous learning rate η_{t-1} and the current loss/gradients for each data point. See the original paper for a detailed description of $\boldsymbol{\phi}$.

c : Daniel et al. (2016) aim to control the learning rate η as to maximally reduce the training loss. Specifically, the cost of a run is quantified as $\min(\frac{1}{k-1} \log(\frac{E_k}{E_1}), 0)$, where E_t is the full batch training loss after t optimization steps. Note that we handle divergence cases by setting the costs of runs that fail to reduce the training loss to 0.

11. In the original paper, Momentum was simply referred to as "SGD".

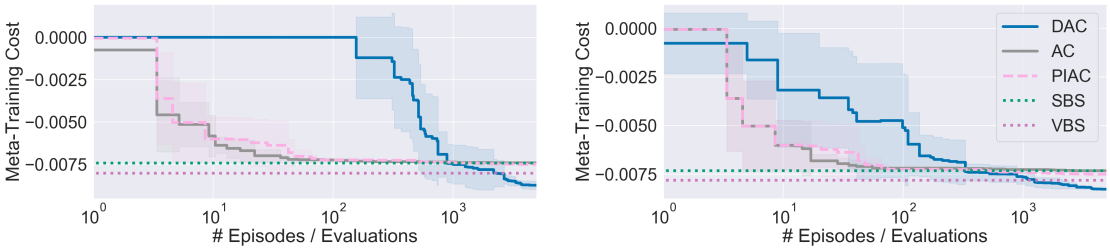
	$\mathcal{A}$	D	L	k	ξ
meta-training	RMSprop	MNIST-small	c-p-c-p-c-r-fc-s (varying # filters)	300-1000	varying
	Momentum	MNIST-small	c-p-c-p-c-r-fc-s (varying # filters)	300-1000	varying
meta-testing	RMSprop	MNIST	c-p-c-p-c-r-fc-s (20-50-200 filters)	2000	fixed
	Momentum	MNIST	c-p-c-p-c-r-fc-s (20-50-200 filters)	2000	fixed
	RMSprop	CIFAR10	c-p-r-c-r-p-c-r-p-c-r-fc-s (32-32-64-64 filters)	6000	fixed
	Momentum	CIFAR10	c-p-r-c-r-p-c-r-p-c-r-fc-s (32-32-64-64 filters)	12000	fixed

Table 3: A summary of the six different DAC setups used in (Daniel et al., 2016). During meta-training, 100 target problem instances are considered, generated by randomly varying D (dataset), L (loss), k (cutoff), and ξ (seed). The meta-testing setups consider a single instance. MNIST-small: To avoid bias towards specific training examples, a randomly varied subset of 6K-30K of the 60K MNIST training examples is used during meta-training. The losses L differ only in the predictive model. All use CNNs, but a different layered architecture (c: same convolution with 3x3 filter, r: ReLU, fc: fully connected, s: softmax). To avoid bias towards specific architectures, the number of filters used is varied randomly in meta-training (in ranges [2-10]-[5-25]-[50-250]).

Solution Method: Daniel et al. (2016) solved this DAC problem by directly optimizing the policy parameters $\boldsymbol{\lambda}$ using the Relative Entropy Policy Search (REPS) policy gradient method. In our experiments, we will also optimize $\boldsymbol{\lambda}$ directly, but instead use Sequential Model-based Algorithm Configuration (SMAC, Hutter et al., 2011). Note that we follow a *DAC by static AC*, instead of a *DAC by RL* approach (see Section 4.2). This decision was motivated by the fact that Daniel et al. (2016) provide too little details about the method and its implementation, to allow us to confidently reproduce the original meta-training pipeline. On the other hand, SMAC is a popular open source (Lindauer et al., 2022) tool for Bayesian optimization that we conjecture to be suitable to reliably and globally optimize $\boldsymbol{\lambda}$ within a reasonable time frame.

Experimental Setup: In our experiments, we used the DACBench implementation of the DAC scenario described above (SGD-DL). Apart from using SMAC instead of REPS, we aimed to maximally replicate the setup used in the original paper. Note that Daniel et al. (2016) actually considered six slightly different scenarios: Two for learning the η-controller for RMSprop/Momentum, resp., and two for testing each of the meta-learned controllers on MNIST/CIFAR10, resp. The differences between these setups are summarized in Table 3. As we did not have access to the original code, replication was restricted by the details disclosed in the original paper.[12] The remaining design choices were mostly made heuristically. Some had to be optimized to obtain similar baseline behavior. Here, we found the use of a sufficiently large mini-batch size (64 at meta-training, 512 at meta-testing), and Xavier weight initialisation, to be particularly important. For meta-training the two η-controllers, we used a meta-training set $I' \sim \mathcal{D}$ of 100 instances and the default parameter settings of SMAC, and optimized $\boldsymbol{\lambda} \in [-10, 10]^5$, using a symmetric log-scale with linear threshold 10^{-6}, for 5000 inner training runs. Each SMAC run took less than 2 CPU-days on our system. To assess meta-training variability, we performed five such runs in parallel,

12. We also contacted Chris Daniel, the first author, but he did not have access to the proprietary code anymore, either, and was thus not able to help us replicate the original setup.



(a) Meta-Training RMSprop

(b) Meta-Training Momentum

Figure 7: Incumbent performance of DAC (SMAC configuring a parametric DAC policy), PIAC, and classical AC when meta-learning learning rate configuration for RMSprop (left) and Momentum (right). Solid lines depict the mean of five independent meta-learning runs and the shaded area the standard deviation. SBS depicts the single best configuration and VBS the oracle configuration selection portfolio across all instances.

selecting the configuration with the highest meta-training performance for meta-testing. For Hydra, we used the same parameters as SMAC, and a maximum portfolio size of 10. Finally, to determine SBS and VBS, we discretized Θ (1000 values, log-scale in $[10^{-5}, 10^{0}]$) and evaluated $c(\boldsymbol{\theta}, i)$ for all (1000×100) combinations.

Results: Figure 7 compares the anytime performance of DAC (SMAC) to that of PIAC (Hydra) and classical AC (SMAC) for RMSprop (left) and Momentum (right). In both cases, DAC's initial performance is worse than its static counterparts. This difference in relative performance is most blatant for RMSprop, where DAC takes over 100 evaluations to find a non-diverging policy (i.e., with negative average cost), while classical AC achieves near SBS performance in that time. PIAC (Hydra) only marginally improves upon classical AC (SMAC) and SBS, and does not attain VBS performance. Despite the slow start, all DAC runs eventually outperform all classical AC and PIAC runs, ultimately attaining a policy that reduces training loss 0.71% (RMSprop) and 0.46% (Momentum) more per step than the VBS on average ($\sim$ 58% and 35% after 650 steps). Figure 8 shows the full batch training loss $L(w_t, D)$ at each optimization step using the meta-learned η-controller that performed best in meta-training, and various static baselines, in each of the four meta-test setups. Overall, the training curves for our baselines look similar to the original, both in terms of absolute and relative performance. An exception are high learning rates. For RMSprop, our curves look quite different, but are similarly chaotic. For momentum, the highest learning rate performs best for us, while the original diverges. On MNIST, both meta-learned controllers (π) clearly outperform the best static baseline, even though the cutoff k is two times higher than the highest cutoff considered during meta-training. This result is similar to that of the original paper, but our learned controller arguably even does better. On CIFAR10, the meta-learned controller (π) performs similar to (RMSprop), or better than (Momentum) the best baseline in the first 1000 update steps, but fails to achieve the best final performance. Here, our results differ from the original, where the final performance was similar (RMSprop) or better (Momentum) than the best static baseline.

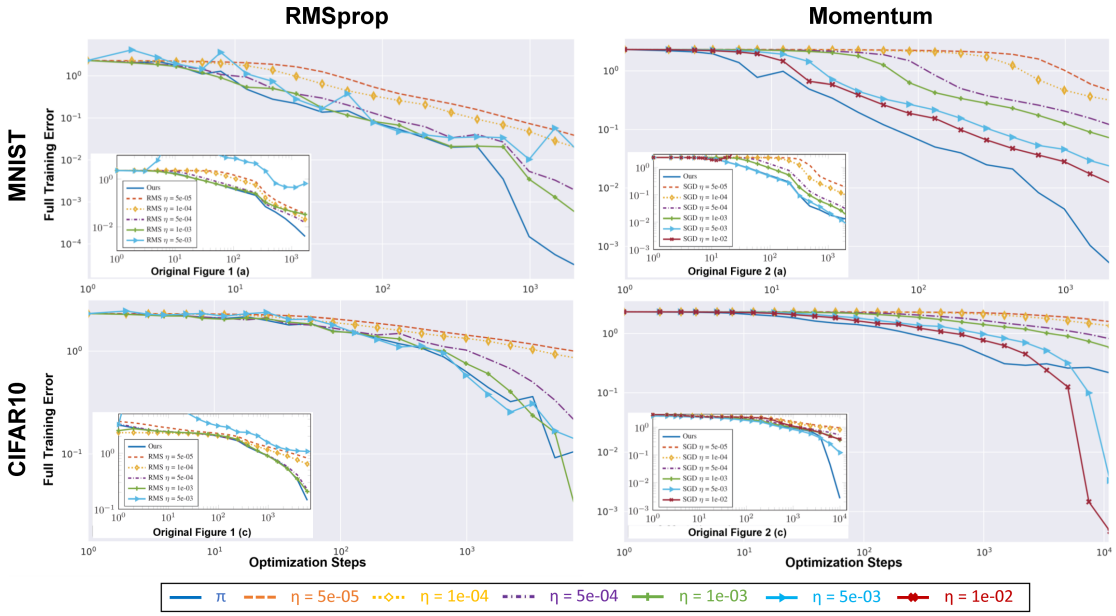


Figure 8: Comparison of learning curves for RMSprop/Momentum using the meta-learned η-controllers (π) vs. several static baselines, on MNIST/CIFAR10. Each dataset/optimizer combination appears in its own sub-figure. For ease of comparison, the corresponding figure in the original paper is shown in the bottom left corner of each sub-figure.

Discussion: The "flavour of AC" prevailing on these scenarios depends on the budget available: For sufficiently large budgets (> 1000 evaluations), DAC confidently outperforms static AC. However, DAC is clearly outperformed by classical AC for smaller budgets. While the best DAC policies are better, arbitrary static policies tend to outperform arbitrary DAC policies for this scenario, e.g., the vast majority of DAC policies diverge for RMSprop. Nonetheless, the poor relative initial performance of DAC is not inherent, and could, e.g., be addressed by using a different initial design that prioritizes static policies ($\pi_{\boldsymbol{\lambda}} : \lambda_k = 0, \forall\, k > 0$). Also note that we cannot compare our meta-training results to those obtained by REPS, since Daniel et al. (2016) did not analyze meta-training. Our meta-testing results, however, validate that the SGD-DL benchmark considers a highly similar setup and that it can be used to learn controllers that perform similarly well as in the original paper. On the other hand, we also observed differences that are unlikely explained by random noise alone. For example, momentum seems to prefer higher learning rates in our experiments, and our meta-learned controller does not transfer as well to higher cutoffs on CIFAR10. Finally, the configurations $\boldsymbol{\lambda}$ we found differ strongly from those reported in the original paper, and using the latter even caused divergence in our experiments. We currently cannot explain those differences, and lacking the original code, further insight can only be gained through trial & error. We emphasize that, in contrast to the original code, our benchmark is publicly available to facilitate future research on DAC.

7. Conclusion

To conclude, we again summarize our main insights and results, and discuss possible future research directions opened up by this work.

7.1 Summary

In this article, we presented the first comprehensive overview of automated Dynamic Algorithm Configuration (DAC), a novel meta-algorithmic framework proposed by Biedenkapp et al. (2020). To this end, we introduced automated DAC as a natural extension of previous research efforts in automated static algorithm configuration and manual DAC. Furthermore, we situated automated DAC in a broader context of AI, discussing how it can be viewed as a form of "semi-automated" programming, as a generalization of existing meta-algorithmic frameworks, and as an automated approach to the design of operator selection and parameter control mechanisms. After formalizing DAC further, we introduced its methodology and showed how prior art can be roughly subdivided in two schools, tackling the problem using *reinforcement learning* and *optimization* methods, respectively. On the empirical side, we presented and extended DACBench, a novel benchmark library for DAC proposed by Eimer et al. (2021b) and showed that DAC can be successfully applied to evolutionary optimization, AI planning, and machine learning. As the first paper, we provided thorough empirical evidence that automated DAC can outperform prior static AC methods. In summary, we found that on all scenarios considered, automated DAC discovered policies that were at least as good as, and typically better than, their static counterparts. Depending on the scenario, this sometimes required less, but usually more (up to $10\times$) computational budget than a state-of-the-art static AC method needed to converge on the same scenario, on average.

7.2 Limitations and Further Research

While these case studies and other previous applications provide a "proof of concept" for automated DAC, we point out that much remains to be done to unlock its full potential, and we hope that this work may serve as a stepping stone for further exploring this promising line of research. In what remains, we will discuss some of the limitations of contemporary work and provide specific directions for future research.

Jointly configuring many parameters: While static approaches are capable of jointly configuring hundreds of parameters, the configuration space in contemporary DAC is typically much smaller, often considering only a single parameter. While the configuration space is smaller, the candidate solution space (i.e., the dynamic configuration policy space) grows exponentially with the number of reconfiguration points, in the worst case, and is thus typically drastically larger than static configuration policy spaces. Although modern techniques from reinforcement learning scale much better than ever before, we still know too little about the internal structure of DAC problems to handle this exploding space of possible policies. For example, not much is known regarding interaction effects of parameters in the DAC setting. If there should be only a few interaction effects between parameters as in static AC (Hutter et al., 2014; Wang et al., 2016), learning several independent policies might be a way forward.

Temporally sparse DAC: Note that not all parameters can/must be reconfigured at every time step. Also, our results suggest that an initial bias towards static configuration policies could improve the anytime performance of DAC in various scenarios. Mixed static and dynamic configuration, and learning "when to reconfigure" (Biedenkapp et al., 2021) therefore present one opportunity to scaling up DAC. Furthermore, we plan to extend the DAC formalism with *partial* reconfiguration to capture intrinsic temporal conditionalities, e.g., a parameter not being used in some execution steps.

Warm-starting DAC: Most prior art derives dynamic configuration policies from scratch, while in many cases good default parameter control mechanisms are known. Beyond strong baselines, these existing policies could also be used to warm-start the automated process. This idea has already been explored by Shala et al. (2020) (see also Section 6.1), but could be extended in various ways. For example, we could learn from an ensemble of teachers to exploit their complementary strengths.

Online DAC: Most prior art performs algorithm configuration offline, i.e., the optimal static/dynamic configuration policy is derived in a dedicated configuration phase proceeding use/test time (see Figure 1).[13] However, when using the target algorithm, more information about the target problem distribution and relative performance of candidate policies becomes available, and *online algorithm configuration* approaches (Fitzgerald, 2021) capable of exploiting this information and transferring experience across test instances, continually refining the policy are an interesting direction of future research.

Better DAC methods: Successful DAC requires more than just computational resources. To apply DAC, a practitioner must make many choices that critically affect not just its efficiency, but also its effectiveness. As a consequence, key ingredients for successful DAC are currently (i) target domain expertise, (ii) DAC methodology expertise, and (iii) trial and error. Note that this conflicts with the main objective of automated DAC, i.e., reducing reliance on human effort and expertise. To address this shortcoming, we need better methods. In particular, as discussed in Section 4.3, we believe that there is a need for dedicated dynamic algorithm configuration packages capable of combining the strengths of the contemporary DAC by reinforcement learning and optimization approaches.

Domain-Expert driven DAC: From working on static AC for more than a decade, we know that a challenge AC poses to users is to specify the inputs, including questions such as: (i) Which instances will reflect future real-world use cases well? (ii) Which parameters are important and should be configured, and using which domain (upper & lower bounds, etc)? (iii) Which metric will accurately quantify the true desirability of a configuration or policy? This hinders the adoption of such meta-algorithmic approaches in practice. To tackle this problem, we envision a new paradigm which is driven by the domain expert and allows for monitoring of the training and deployment performance, and for live adjustments of training distributions, configuration spaces and performance metrics by the domain expert. Likewise, we would like to enable experts to express priors over the policies they would expect to work well, extending similar work in static AC (Hvarfner et al., 2022). Finally, we would like domain experts to not only be able to steer DAC, but to also gain new and

13. As discussed in Section 2.2.3, we do not regard the majority of previous "online learning approaches" to parameter control as prior art in *automating* DAC.

deeper insights from the automated search process, similar to various existing methods that capture the importance of hyperparameters in static AC (Hutter et al., 2014; Biedenkapp et al., 2017, 2018; van Rijn & Hutter, 2018; Probst et al., 2019).

Extending DAC Benchmarks: To stay relevant, these future directions will also have to be reflected in a set of contemporary DAC benchmarks, such as in DACBench, alongside continuing work on further expanding the scope of existing benchmarks. While DACBench covers a range of domains, some like SAT or MIP, which are commonly used in AC, are absent at the moment. Partnering with domain experts could help broaden the scope of DACBench and thus DAC in general. Beyond real-world benchmarks, there is also a need for additional "toy" benchmarks that permit efficient evaluation of DAC methods, something especially crucial in (i) the early stages of developing new methods and (ii) enabling meta-algoritmics to be applied to DAC itself. Finally, prior art, our own work included, rarely compares different DAC methods. To facilitate this, we need more than just benchmarks, we need a library of DAC methods and standard protocols to compare them.

Acknowledgments

All authors acknowledge funding by the Robert Bosch GmbH. Theresa Eimer and Marius Lindauer acknowledge funding by the German Research Foundation (DFG) under LI 2801/4-1. We thank Maximilian Reimer, Rishan Senanayake, Göktuğ Karakaşlı, Nguyen Dang, Diederick Vermetten, Jacob de Nobel and Carolin Benjamins for their contributions to DACBench, and Carola Doerr for the many discussions on related work and problem formulation.

References

Adriaensen, S., & Nowé, A. (2016). Towards a white box approach to automated algorithm design.. In Kambhampati, S. (Ed.), *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, pp. 554–560.

Adriaensen, S. (2018). *On the Semi-automated Design of Reusable Heuristics*. Ph.D. thesis, Vrije Universiteit Brussel.

Aleti, A., & Moser, I. (2016). A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Comput. Surv.*, *49*(3), 1–35.

Almeida, D., Winter, C., Tang, J., & Zaremba, W. (2021). A generalizable approach to learning optimizers. *arXiv preprint arXiv:2106.00958*, *[cs.LG]*.

Andersson, M., Bandaru, S., & Ng, A. H. (2016). Tuning of multiple parameter sets in evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 533–540.

Andrychowicz, M., Denil, M., Colmenarejo, S. G., Hoffman, M., Pfau, D., Schaul, T., & de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I., & Garnett, R. (Eds.), *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16)*, pp. 3981–3989. Curran Associates.

Ansótegui, C., Malitsky, Y., Sellmann, M., & Tierney, K. (2015). Model-based genetic algorithms for algorithm configuration. In Yang, Q., & Wooldridge, M. (Eds.), *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pp. 733–739.

Ansótegui, C., Pon, J., Sellmann, M., & Tierney, K. (2021). PyDGGA: Distributed GGA for automatic configuration. In Li, C., & Manyà, F. (Eds.), *Theory and Applications of Satisfiability Testing - SAT*, Vol. 12831 of *Lecture Notes in Computer Science*, pp. 11–20. Springer.

Ansótegui, C., Sellmann, M., & Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In Gent, I. (Ed.), *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP'09)*, Vol. 5732 of *Lecture Notes in Computer Science*, pp. 142–157. Springer.

Ansótegui, C., Pon, J., Sellmann, M., & Tierney, K. (2017). Reactive dialectic search portfolios for MaxSAT. In S.Singh, & Markovitch, S. (Eds.), *Proceedings of the Thirty-First Conference on Artificial Intelligence (AAAI'17)*, pp. 765–772. AAAI Press.

Barozet, A., Molloy, K., Vaisset, M., Siméon, T., & Cortés, J. (2020). A reinforcement-learning-based approach to enhance exhaustive protein loop sampling. *Bioinform.*, *36*(4), 1099–1106.

Battiti, R., & Campigotto, P. (2012). An investigation of reinforcement learning for reactive search optimization. In Hamadi, Y., Monfroy, E., & Saubion, F. (Eds.), *Autonomous Search*, pp. 131–160. Springer.

Battiti, R., Brunato, M., & Mascia, F. (2008). *Reactive search and intelligent optimization*, Vol. 45. Springer Science & Business Media.

Battiti, R., & Tecchiolli, G. (1994). The reactive tabu search. *ORSA journal on computing*, *6*(2), 126–140.

Baydin, A., Cornish, R., Rubio, D., Schmidt, M., & Wood, F. (2018). Online learning rate adaption with hypergradient descent. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*. Published online: `iclr.cc`.

Bello, I., Zoph, B., Vasudevan, V., & Le, Q. V. (2017). Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, pp. 459–468. PMLR.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pp. 437–478. Springer.

Bhatia, A., Svegliato, J., & Zilberstein, S. (2021). Tuning the hyperparameters of anytime planning: A deep reinforcement learning approach. In *ICAPS 2021 Workshop on Heuristics and Search for Domain-independent Planning*.

Biedenkapp, A., Bozkurt, H. F., Eimer, T., Hutter, F., & Lindauer, M. (2020). Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In Lang, J., Giacomo, G. D., Dilkina, B., & Milano, M. (Eds.), *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*, pp. 427–434.

Biedenkapp, A., Dang, N., Krejca, M. S., Hutter, F., & Doerr, C. (2022). Theory-inspired parameter control benchmarks for dynamic algorithm configuration. In Fieldsend, J. (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'22)*. ACM.

Biedenkapp, A., Lindauer, M., Eggensperger, K., Fawcett, C., Hoos, H., & Hutter, F. (2017). Efficient parameter importance analysis via ablation with surrogates. In S.Singh, & Markovitch, S. (Eds.), *Proceedings of the Thirty-First Conference on Artificial Intelligence (AAAI'17)*, pp. 773–779. AAAI Press.

Biedenkapp, A., Marben, J., Lindauer, M., & Hutter, F. (2018). CAVE: Configuration assessment, visualization and evaluation. In Battiti, R., Brunato, M., Kotsireas, I., & Pardalos, P. (Eds.), *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*, Lecture Notes in Computer Science. Springer.

Biedenkapp, A., Rajan, R., Hutter, F., & Lindauer, M. (2021). TempoRL: Learning when to act. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*.

Birattari, M., Stützle, T., Paquete, L., & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In Langdon, W., Cantu-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M., Schultz, A., Miller, J., Burke, E., & Jonoska, N. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, pp. 11–18. Morgan Kaufmann Publishers.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Frechétte, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., & Vanschoren, J. (2016). ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, *237*, 41–58.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1), 5–33.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*, *[cs.LG]*.

Carchrae, T., & Beck, J. (2004). Low-knowledge algorithm control. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, p. 49–54. AAAI Press.

Chen, F., Gao, Y., qian Chen, Z., & fu Chen, S. (2005). SCGA: Controlling genetic algorithms with sarsa(0). In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Vol. 1, pp. 1177–1183.

Chen, T., Chen, X., Chen, W., Heaton, H., Liu, J., Wang, Z., & Yin, W. (2021). Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*, *[cs.LG]*.

Chrabaszcz, P., Loshchilov, I., & Hutter, F. (2018). Back to basics: Benchmarking canonical evolution strategies for playing atari. In Lang, J. (Ed.), *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*, pp. 1419–1426. ijcai.org.

Daniel, C., Taylor, J., & Nowozin, S. (2016). Learning step size controllers for robust neural network training. In Schuurmans, D., & Wellman, M. (Eds.), *Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16)*. AAAI Press.

Doerr, B. (2019). Analyzing randomized search heuristics via stochastic domination. *Theoretical Computer Science*, *773*, 115–137.

Doerr, B., & Doerr, C. (2020). Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In Doerr, B., & Neumann, F. (Eds.), *Theory of Evolutionary Computation*, pp. 271–321. Springer.

Doerr, C., Wang, H., Ye, F., van Rijn, S., & Bäck, T. (2018). IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *arXiv preprint arXiv:1810.05281*, *[cs.NE]*.

Drake, J. H., Kheiri, A., Özcan, E., & Burke, E. K. (2020). Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, *285*(2), 405–428.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., & Leyton-Brown, K. (2013). Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13)*.

Eggensperger, K., Lindauer, M., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2018). Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, *107*(1), 15–41.

Eiben, A., Horvath, M., Kowalczyk, W., & Schut, M. (2006). Reinforcement learning for online control of evolutionary algorithms. In *International Workshop on Engineering Self-Organising Applications*, pp. 151–160. Springer.

Eimer, T., Biedenkapp, A., Hutter, F., & Lindauer, M. (2021a). Self-paced context evaluation for contextual reinforcement learning. In Meila, M., & Zhang, T. (Eds.), *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, Vol. 139 of *Proceedings of Machine Learning Research*, pp. 2948–2958. PMLR.

Eimer, T., Biedenkapp, A., Reimer, M., Adriaensen, S., Hutter, F., & Lindauer, M. (2021b). DACBench: A benchmark library for dynamic algorithm configuration. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI'21)*. ijcai.org.

Falkner, S., Klein, A., & Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In Dy, J., & Krause, A. (Eds.), *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, Vol. 80, pp. 1437–1446. Proceedings of Machine Learning Research.

Fawcett, C., Vallati, M., Hutter, F., Hoffmann, J., Hoos, H., & Leyton-Brown, K. (2014). Improved features for runtime prediction of domain-independent planners. In Chien, S., Minh, D., Fern, A., & Ruml, W. (Eds.), *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS-14)*, pp. 355–359. AAAI.

Feurer, M., & Hutter, F. (2019). Hyperparameter optimization. In Hutter, F., Kotthoff, L., & Vanschoren, J. (Eds.), *Automated Machine Learning: Methods, Systems, Challenges*, pp. 3–38. Springer. Available for free at http://automl.org/book.

Fialho, A., Da Costa, L., Schoenauer, M., & Sebag, M. (2010). Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, *60*(1), 25–64.

Fink, E. (1998). How to solve it automatically: Selection among problem solving methods.. In *AIPS*, pp. 128–136.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D., & Teh, Y. W. (Eds.), *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135. PMLR.

Fitzgerald, T. (2021). *Real-time algorithm configuration*. Ph.D. thesis, University College Cork.

Fu, J. (2016). Deep q-networks for accelerating the training of deep neural networks. *arXiv preprint arXiv:1606.01467, [cs.LG]*.

Gagliolo, M., & Schmidhuber, J. (2006). Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence*, *47*(3-4), 295–328.

Gaspero, L. D., & Urli, T. (2012). Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In Hamadi, Y., & Schoenauer, M. (Eds.), *Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION'12)*, Vol. 7219 of *Lecture Notes in Computer Science*, pp. 384–389. Springer.

Getzelman, G., & Balaprakash, P. (2021). Learning to switch optimizers for quadratic programming. In *Asian Conference on Machine Learning*, pp. 1553–1568. PMLR.

Gomes, C., & Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, *126*(1-2), 43–62.

Gomoluch, P., Alrajeh, D., & Russo, A. (2019). Learning classical planning strategies with policy gradient. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 29, pp. 637–645.

Gomoluch, P., Alrajeh, D., Russo, A., & Bucchiarone, A. (2020). Learning neural search policies for classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 30, pp. 522–530.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*, *[cs.NE]*.

Hall, G. T., Oliveto, P. S., & Sudholt, D. (2019). On the impact of the cutoff time on the performance of algorithm configurators. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 907–915.

Hall, G. T., Oliveto, P. S., & Sudholt, D. (2020). Fast perturbative algorithm configurators. In *International Conference on Parallel Problem Solving from Nature*, pp. 19–32. Springer.

Hallak, A., Di Castro, D., & Mannor, S. (2015). Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, *[stat.ML]*.

Hansen, N., Finck, S., Ros, R., & Auger, A. (2009). Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research report RR-6829, INRIA.

Hansen, N., Müller, S. D., & Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computing*, *11*(1), 1–18.

Hansen, N., & Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pp. 312–317. IEEE.

Hansen, S. (2016). Using deep q-learning to control optimization hyperparameters. *arXiv preprint arXiv:1602.04062*, *[math.OC]*.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Helmert, M., Röger, G., & Karpas, E. (2011). Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS-2011 Workshop on Planning and Learning (PAL)*, pp. 28–35.

Helmert, M. (2004). A planning heuristic based on causal graph analysis. In Zilberstein, S., Koehler, J., & Koenig, S. (Eds.), *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-04)*, pp. 161–170. AAAI Press.

Helmert, M., & Geffner, H. (2008). Unifying the causal graph and additive heuristics. In Rintanen, J., Nebel, B., Beck, J. C., & Hansen, E. (Eds.), *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-08)*, pp. 140–147. AAAI Press.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. In McIlraith, S., & Weinberger, K. (Eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'18)*. AAAI Press.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Hoos, H. (2012). Programming by optimization. *Communications of the ACM*, *55*(2), 70–80.

Hoos, H., Kaminski, R., Lindauer, M., & Schaub, T. (2015). aspeed: Solver scheduling via answer set programming. *Theory and Practice of Logic Programming*, *15*, 117–142.

Huberman, B., Lukose, R., & Hogg, T. (1997). An economic approach to hard computational problems. *Science*, *275*, 51–54.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. In Lodi, A., Milano, M., & Toth, P. (Eds.), *Proceedings of the Seventh International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR'10)*, Vol. 6140 of *Lecture Notes in Computer Science*, pp. 186–202. Springer.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In Coello, C. (Ed.), *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, Vol. 6683 of *Lecture Notes in Computer Science*, pp. 507–523. Springer.

Hutter, F., Hoos, H., & Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In Xing, E., & Jebara, T. (Eds.), *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, pp. 754–762. Omnipress.

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

Hutter, F., López-Ibánez, M., Fawcett, C., Lindauer, M., Hoos, H., Leyton-Brown, K., & Stützle, T. (2014). AClib: a benchmark library for algorithm configuration. In Pardalos, P., & Resende, M. (Eds.), *Proceedings of the Eighth International Conference on Learning and Intelligent Optimization (LION'14)*, Lecture Notes in Computer Science, pp. 36–40. Springer.

Hvarfner, C., Stoll, D., Souza, A., Nardi, L., Lindauer, M., & Hutter, F. (2022). PiBO: Augmenting Acquisition Functions with User Beliefs for Bayesian Optimization. In *International Conference on Learning Representations*.

Ichnowski, J., Jain, P., Stellato, B., Banjac, G., Luo, M., Borrelli, F., Gonzalez, J. E., Stoica, I., & Goldberg, K. (2021). Accelerating quadratic optimization with reinforcement learning. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural networks*, *1*(4), 295–307.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., & Kavukcuoglu, K. (2017).

Population based training of neural networks. *arXiv preprint arXiv:1711.09846, [cs.LG]*.

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2011). Algorithm selection and scheduling. In Lee, J. (Ed.), *Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP'11)*, Vol. 6876 of *Lecture Notes in Computer Science*, pp. 454–469. Springer.

Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). ISAC - instance-specific algorithm configuration. In Coelho, H., Studer, R., & Wooldridge, M. (Eds.), *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI'10)*, pp. 751–756. IOS Press.

Kadioglu, S., Sellmann, M., & Wagner, M. (2017). Learning a reactive restart strategy to improve stochastic search. In *International Conference on Learning and Intelligent Optimization*, pp. 109–123. Springer.

Karafotias, G., Eiben, A. E., & Hoogendoorn, M. (2014). Generic parameter control with reinforcement learning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 1319–1326.

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

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*. Published online: `iclr.cc`.

Kleinberg, R., Leyton-Brown, K., & Lucier, B. (2017). Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees.. In Sierra, C. (Ed.), *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pp. 2023–2031.

Klink, P., D'Eramo, C., Peters, J., & Pajarinen, J. (2020). Self-paced deep reinforcement learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS*.

Kool, W., van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems!. In *International Conference on Learning Representations*.

Kotthoff, L. (2014). Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, *35*(3), 48–60.

Koza, J. R. (1992). *Genetic programming*, Vol. 4. MIT Press.

Lagoudakis, M., & Littman, M. (2001). Learning to select branching rules in the DPLL procedure for satisfiability. *Electronic Notes in Discrete Mathematics*, *9*, 344–359.

Lagoudakis, M. G., & Littman, M. L. (2000). Algorithm selection using reinforcement learning.. In *ICML*, pp. 511–518.

Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., & Hutter, M. (2020). Learning quadrupedal locomotion over challenging terrain. *Science in Robotics*, *5*.

Levine, S., & Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., & Weinberger, K. (Eds.), *Proceedings of the 27th International Conference on Advances in Neural Information Processing Systems (NeurIPS'14)*, pp. 1071–1079. Curran Associates.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., & Shoham, Y. (2003). A portfolio approach to algorithm selection. In *IJCAI*, Vol. 3, pp. 1542–1543.

Li, K., & Malik, J. (2017). Learning to optimize. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, *18*(185), 1–52.

Lindauer, M., Bergdoll, D., & Hutter, F. (2016). An empirical study of per-instance algorithm scheduling. In Festa, P., Sellmann, M., & Vanschoren, J. (Eds.), *Proceedings of the Tenth International Conference on Learning and Intelligent Optimization (LION'16)*, Lecture Notes in Computer Science. Springer. to appear.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., & Hutter, F. (2022). SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research (JMLR) – MLOSS*, *23*(54), 1–9.

Lobo, F., Lima, C. F., & Michalewicz, Z. (2007). *Parameter setting in evolutionary algorithms*, Vol. 54. Springer Science & Business Media.

López-Ibáñez, M., Dubois-Lacoste, J., Caceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

López-Ibánez, M., & Stützle, T. (2014). Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, *235*(3), 569–582.

Loshchilov, I., & Hutter, F. (2017). SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

Lv, K., Jiang, S., & Li, J. (2017). Learning gradient descent: Better generalization and longer horizons. In *International Conference on Machine Learning*, pp. 2247–2255. PMLR.

Maclaurin, D., Duvenaud, D., & Adams, R. (2015). Gradient-based Hyperparameter Optimization through Reversible Learning. In Bach, F., & Blei, D. (Eds.), *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, Vol. 37, pp. 2113–2122. Omnipress.

Majid, A. (2021). Deep reinforcement learning versus evolution strategies: A comparative survey. *arXiv preprint arXiv:2110.01411*, *[cs.LG]*.

Manna, Z., & Waldinger, R. (1980). A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *2*(1), 90–121.

Mannor, S., Rubinstein, R. Y., & Gat, Y. (2003). The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 512–519.

Metz, L., Maheswaranathan, N., Freeman, C., Poole, B., & Sohl-Dickstein, J. (2020). Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243, [cs.LG]*.

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., & Sohl-Dickstein, J. (2019). Understanding and correcting pathologies in the training of learned optimizers. In Chaudhuri, K., & Salakhutdinov, R. (Eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, Vol. 97. Proceedings of Machine Learning Research.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Moulines, E., & Bach, F. R. (2011). Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., & Weinberger, K. (Eds.), *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'11)*, pp. 451–459. Curran Associates.

Muller, S. D., Schraudolph, N. N., & Koumoutsakos, P. D. (2002). Step size adaptation in evolution strategies using reinforcement learning. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, Vol. 1, pp. 151–156. IEEE.

Nguyen, M. H., Grinsztajn, N., Guyon, I., & Sun-Hosoya, L. (2021). MetaREVEAL: RL-based meta-learning from learning curves. In *Workshop on Interactive Adaptive Learning*.

Papadimitriou, C. H. (1994). On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and system Sciences*, *48*(3), 498–532.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., & Garnett, R. (Eds.), *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*. Curran Associates.

Pettinger, J., & Everson, R. (2002). Controlling genetic algorithms with reinforcement learning. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pp. 692–692.

Prestwich, S. (2008). Tuning local search by average-reward reinforcement learning. In Maniezzo, V., Battiti, R., & Watson, J. (Eds.), *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*, Vol. 5313 of *Lecture Notes in Computer Science*, pp. 192–205. Springer.

Probst, P., Boulesteix, A., & Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, *20*(53), 1–32.

Pushak, Y., & Hoos, H. (2020). Golden parameter search: exploiting structure to quickly configure parameters in parallel. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 245–253.

Rice, J. (1976). The algorithm selection problem. *Advances in Computers*, *15*, 65–118.

Richter, S., & Helmert, M. (2009). Preferred operators and deferred evaluation in satisficing planning. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS-09)*, pp. 273–280. AAAI Press.

Richter, S., Helmert, M., & Westphal, M. (2008). Landmarks revisited. In Fox, D., & Gomes, C. P. (Eds.), *Proceedings of the Twenty-third Conference on Artificial Intelligence (AAAI'08)*, pp. 975–982. AAAI Press.

Röger, G., & Helmert, M. (2010). The more, the merrier: Combining heuristic estimators for satisficing planning. In Brafman, R., Geffner, H., Hoffmann, J., & Kautz, H. (Eds.), *Working notes of the Twenty-first International Conference on Automated Planning and Scheduling (ICAPS-10), Workshop on Planning and Learning.*, pp. 246–249.

Sae-Dan, W., Kessaci, M.-E., Veerapen, N., & Jourdan, L. (2020). Time-dependent automatic parameter configuration of a local search algorithm. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, p. 1898–1905.

Sakurai, Y., Takada, K., Kawabe, T., & Tsuruta, S. (2010). A method to control parameters of evolutionary algorithms by using reinforcement learning. In Yétongnon, K., Dipanda, A., & Chbeir, R. (Eds.), *Proceedings of Sixth International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*, pp. 74–79. IEEE Computer Society.

Salimans, T., Ho, J., Chen, X., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, *[stat.ML]*.

Schaul, T., Zhang, S., & LeCun, Y. (2013). No More Pesky Learning Rates. In Dasgupta, S., & McAllester, D. (Eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*. Omnipress.

Seipp, J., Sievers, S., Helmert, M., & Hutter, F. (2015). Automatic configuration of sequential planning portfolios. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*. AAAI Press.

Seipp, J., Sievers, S., & Hutter, F. (2014). Fast downward SMAC.. Planner abstract, IPC 2014 Planning and Learning Track.

Senior, A., Heigold, G., Ranzato, M., & Yang, K. (2013). An empirical study of learning rates in deep neural networks for speech recognition. In *Proc. of ICASSP*.

Shala, G., Biedenkapp, A., Awad, N., Adriaensen, S., Lindauer, M., & Hutter, F. (2020). Learning step-size adaptation in CMA-ES. In Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M., & Trautmann, H. (Eds.), *Proceedings of the Sixteenth*

International Conference on Parallel Problem Solving from Nature (PPSN'20), Lecture Notes in Computer Science, pp. 691–706. Springer.

Sharma, M., Komninos, A., López-Ibáñez, M., & Kazakov, D. (2019). Deep reinforcement learning based parameter control in differential evolution. In López-Ibáñez, M. (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 709–717. ACM.

Sievers, S., Katz, M., Sohrabi, S., Samulowitz, H., & Ferber, P. (2019). Deep learning for cost-optimal planning: Task-dependent planner selection. In Hentenryck, P. V., & Zhou, Z. (Eds.), *Proceedings of the Thirty-Third Conference on Artificial Intelligence (AAAI'19)*, pp. 7715–7723. AAAI Press.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Smith, L. N. (2017). Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pp. 464–472. IEEE.

Snoek, J., Larochelle, H., & Adams, R. (2012). Practical Bayesian optimization of machine learning algorithms. In Bartlett, P., Pereira, F., Burges, C., Bottou, L., & Weinberger, K. (Eds.), *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*, pp. 2960–2968. Curran Associates.

Speck, D., Biedenkapp, A., Hutter, F., Mattmüller, R., & Lindauer, M. (2021). Learning heuristic selection with dynamic algorithm configuration. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21)*.

Stanley, K. O., Clune, J., Lehman, J., & Miikkulainen, R. (2021). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, *1*, 24–35.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*, 99–127.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Szita, I., & Lörincz, A. (2006). Learning tetris using the noisy cross-entropy method. *Neural computation*, *18*(12), 2936–2941.

Tieleman, T., Hinton, G., et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, *4*(2), 26–31.

van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In Schuurmans, D., & Wellman, M. (Eds.), *Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16)*. AAAI Press.

van Rijn, J., & Hutter, F. (2018). Hyperparameter importance across datasets. In Guo, Y., & F.Farooq (Eds.), *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 2367–2376. ACM Press.

van Rijn, S., Doerr, C., & Bäck, T. (2018). Towards an adaptive CMA-ES configurator. In Auger, A., Fonseca, C. M., Lourenço, N., Machado, P., Paquete, L., & Whitley, L. D. (Eds.), *Proceedings of the 15th International Conference on Parallel Problem Solving from Nature (PPSN'18)*, Vol. 11101 of *Lecture Notes in Computer Science*, pp. 54–65. Springer.

Vermetten, D., van Rijn, S., Bäck, T., & Doerr, C. (2019). Online selection of CMA-ES variants. In *Proc. of GECCO*, pp. 951–959. ACM.

Wang, Z., Hutter, F., Zoghi, M., Matheson, D., & de Feitas, N. (2016). Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, *55*, 361–387.

Weisz, G., György, A., & Szepesvári, C. (2019). CapsAndRuns: An improved method for approximately optimal algorithm configuration. In Chaudhuri, K., & Salakhutdinov, R. (Eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, Vol. 97, pp. 6707–6715. Proceedings of Machine Learning Research.

Wessing, S., Preuss, M., & Rudolph, G. (2011). When parameter tuning actually is parameter control. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 821–828.

Wolpert, D. H., & Macready, W. G. (1995). No free lunch theorems for search. Tech. rep. SFI-TR-95-02-010, Santa Fe Institute.

Xu, C., Qin, T., Wang, G., & Liu, T.-Y. (2017). Reinforcement learning for learning rate control. *arXiv preprint arXiv:1705.11159*, *[cs.LG]*.

Xu, L., Hoos, H., & Leyton-Brown, K. (2010). Hydra: Automatically configuring algorithms for portfolio-based selection. In Fox, M., & Poole, D. (Eds.), *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, pp. 210–216. AAAI Press.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2008). SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, *32*, 565–606.

Xu, Z., Dai, A. M., Kemp, J., & Metz, L. (2019). Learning an adaptive learning rate schedule. *arXiv preprint arXiv:1909.09712*, *[cs.LG]*.

Part III

Dynamic Algorithm Configuration: Case Studies

CHAPTER 7

Learning Step-Size Adaptation in CMA-ES

The content of this chapter has been published as:

> G. Shala, A. Biedenkapp, N. Awad, S. Adriaensen, M. Lindauer, and F. Hutter (2020). "Learning Step-Size Adaptation in CMA-ES". in: *Proceedings of the Sixteenth International Conference on Parallel Problem Solving from Nature (PPSN'20)*. Ed. by T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, and H. Trautmann. Lecture Notes in Computer Science. Springer, pp. 691–706.

Project Idea. The idea was proposed by Frank Hutter. André Biedenkapp proposed the extension to GPS for improved sample efficiency. Gresa Shala proposed the initial state and action spaces, building on work by Ke Li and Jitendra Malik. Gresa Shala and André Biedenkapp jointly refined these spaces. Marius Lindauer and Noor Awad proposed the set of benchmarks for evaluation.

Implementation and experimentation. Gresa Shala led the implementation. André Biedenkapp, Noor Awad and Marius Lindauer proposed the experiment design to study the generality of the learned policies. André Biedenkapp proposed to use the performance metric and Steven Adriaensen demonstrated that this metric can be interpreted statistically as it has a correspondence with the "sum of ranks" statistic of the Wilcoxon rank-sum test. Experiments were conducted by Gresa Shala under the supervision of André Biedenkapp.

Paper writing. A first draft of the paper was prepared by André Biedenkapp. Gresa Shala and André Biedenkapp jointly wrote the experiments section. Steven Adriaensen provided the final related work section. Noor Awad aided in writing the background section. André Biedenkapp revised this draft which was further revised and edited by Marius Lindauer and Frank Hutter.

Learning Step-Size Adaptation in CMA-ES

Gresa Shala[1⋆], André Biedenkapp[1⋆], Noor Awad[1], Steven Adriaensen[1], Marius Lindauer[2], and Frank Hutter[1,3]

[1] University of Freiburg, Germany
{shalag, biedenka, awad, adriaens, fh}@cs.uni-freiburg.de
[2] Leibniz University Hannover, Germany
lindauer@tnt.uni-hannover.de
[3] Bosch Center for Artificial Intelligence

Abstract. An algorithm's parameter setting often affects its ability to solve a given problem, e.g., population-size, mutation-rate or crossover-rate of an evolutionary algorithm. Furthermore, some parameters have to be adjusted dynamically, such as lowering the mutation-strength over time. While hand-crafted heuristics offer a way to fine-tune and dynamically configure these parameters, their design is tedious, time-consuming and typically involves analyzing the algorithm's behavior on simple problems that may not be representative for those that arise in practice. In this paper, we show that formulating dynamic algorithm configuration as a reinforcement learning problem allows us to automatically learn policies that can dynamically configure the mutation step-size parameter of Covariance Matrix Adaptation Evolution Strategy (CMA-ES). We evaluate our approach on a wide range of black-box optimization problems, and show that (i) learning step-size policies has the potential to improve the performance of CMA-ES; (ii) learned step-size policies can outperform the default Cumulative Step-Size Adaptation of CMA-ES; and transferring the policies to (iii) different function classes and to (iv) higher dimensions is also possible.

Keywords: Evolutionary Algorithms · Reinforcement Learning · Algorithm Configuration.

1 Introduction

Designing algorithms requires careful design of multiple components. Having the foresight of how these components will interact for all possible applications is an infeasible task. Therefore, instead of hard-wiring algorithms, human developers often expose difficult design decisions as parameters of the algorithm [26]. To make the algorithm usable off-the-shelf, they provide a default configuration that is a myopic compromise for different use-cases and often leads to sub-optimal performance on new applications.

Automated algorithm configuration can alleviate users from the burden of having to manually configure an algorithm and exceeds human performance in

⋆ Equal Contribution

a wide variety of domains [7, 27, 43, 42, 5, 29]. One shortcoming, however, is that the learned configuration is static. In practice, many algorithms are of an iterative nature and might require different parameter configurations at different stages of their execution. In evolutionary algorithms this kind of "parameter control" is often achieved through so-called self-adaptive mechanisms [9, 34, 2]. Based on some statistics of the algorithm's behavior, self-adaptation adjusts the parameter on-the-fly and thereby directly influences the algorithm's execution.

Similarly in the well-known CMA-ES [19] the step-size is adapted based on the observed evolution path by a handcrafted heuristic, called CSA [25]. Through this step-size control, CMA-ES is able to avoid premature convergence of the population [21]. However, designing heuristics to adapt not only over a time-horizon but also to the task at hand is more difficult than to simply expose the parameters and configure them at every step.

In this work, we aim to strike a balance between self-adaptive mechanisms and automated algorithm configuration by making use of dynamic algorithm configuration (DAC) [10]. Instead of only learning the optimal *initial* step-size and adapting that by a handcrafted heuristic throughout the run of the algorithm, we learn a DAC policy in a fully automatic and data-driven way that determines how the step-size should be adjusted during the CMA-ES execution.

To learn DAC policies, we make use of guided policy search (GPS) [37], a commonly used reinforcement learning (RL) technique, originating from the robotics community, capable of learning complex non-linear policies from fairly few trials. Our choice for this particular method was motivated by its capability to learn simple first-order optimizers from scratch [39]. An appealing feature of GPS is that it allows us to employ known adaptation schemes as teacher mechanism to warm-start the search. This learning paradigm allows the agent to simply imitate the teacher if it was already optimal for a specific problem, while learning to do better in areas where the teacher struggled to perform well.

We study the potential of this DAC approach to step-size adaptation in CMA-ES for a variety of black-box optimization problems. One important open question so far is how such data-driven approaches can generalize to different settings (e.g., longer optimization runs, higher-dimensional problems or different problem classes) that were not observed during training. More specifically, our contributions are:

1. We address the problem of learning step-size control for CMA-ES from a reinforcement learning perspective;
2. We propose how to model the state space, action space and reward function;
3. To use guided policy search for learning a DAC policy in efficient way, we propose to use a strong teacher guidance.
4. We empirically demonstrate that our learned DAC policies are able to outperform CMA-ES' handcrafted step-size adaptation;
5. We demonstrate the generality of our DAC approach by transferring the learned policies to (i) functions of higher dimensions, (ii) unseen test function and (iii) to a certain degree to longer optimization trajectories.

2 Related Work

Parameter Control using Reinforcement Learning The potential generality of DAC via RL is widely recognized [33, 1, 10] and RL has been applied to various specific parameter control settings [45, 46, 12, 15, 49, 8, 18, 33, 51]. However, RL covers a wide variety of techniques, and our methodology differs from prior-art in the area, in two important ways. First, GPS learns configuration policies *offline*, while most previous research considers the online setting. They attempt to learn how to adapt the parameters of an algorithm "while it is being used", i.e. without separate training phase. While desirable, online learning introduces a challenging exploration-exploitation trade-off. Also, experience is typically not transferred across runs, similar to hand-crafted adaptation mechanisms. That being said, prior-art considering the offline setting does exist, e.g., Battiti et al. [8] for local search SAT solvers and Sharma et al. [51] for EA. Second, GPS belongs to the family of *policy search* methods, which are often able to handle partially observable state spaces and continuous actions spaces better than previously used value-based RL methods.

Black-Box Dynamic Algorithm Configuration In a sense, our methodology more closely resembles static algorithm configuration (AC) than traditional RL approaches. We represent configuration policies as a neural network; and as in AC, train it offline. Instead of GPS, black-box optimizers, e.g. ES, could also be used to optimize these weights [31, 50, 17]. In fact, prior-art exists that performs DAC using static AC methods [35, 3, 6, 32]. A limitation of these "black-box" approaches is that they are unaware of the dynamic nature of the problem [1], e.g. which configurations where used at each time step, and how this affected execution. As a consequence, they are not sample-efficient, and practical applications with long trajectories are forced to consider restrictive policy spaces.

Learning to Optimize in Machine Learning Learning to Learn (L2L) is a form of meta-learning, aiming to use machine learning methods to learn better machine learning systems [53]. Research in the area has recently surged in popularity, resulting in various applications learning better neural network architectures [57, 41], hyper-parameters [44], initialization [16], and optimizers [4, 13, 11, 39, 56, 14]. As we are learning a component of an optimizer, our work is closely related to *Learning to Optimize* (L2O). Note that most L2O research [4, 39, 56, 14] focuses on learning better gradient-based methods (e.g. Adam [36]), as these are most commonly used to optimize neural networks. Notable exceptions are L2O applications to single-point [13] and multi-point [11] black-box optimization. One L2O approach [13, 4, 11] models iterative optimizers as a kind of recurrent neural network and trains it in a fully supervised fashion. More general RL methods have also been used in L2O, e.g. REPS [14], PPO [56], and GPS [39]. In this work, we apply GPS in a similar way. The main difference is that, instead of learning a simple first-order optimization method from scratch, we apply this method to dynamically configure a single parameter (step-size) in a state-of-the-art derivative-free method (CMA-ES).

3 Background on CMA-ES

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [24] is an evolutionary algorithm optimizing a continuous black-box function $f : \mathbb{R}^n \to \mathbb{R}$ by sampling individuals from a non-stationary multivariate normal search distribution $\mathcal{N}\left(m^{(g)}, {\sigma^{(g)}}^2 C^{(g)}\right)$, with mean $m^{(g)}$ (center), step-size $\sigma^{(g)}$ (scale) and covariance matrix $C^{(g)}$ (shape).

Initially, $C^{(0)} = I$ (identity matrix) and $m^{(0)}$, $\sigma^{(0)}$ are provided by the user. The algorithm then iteratively updates this search distribution to increase the probability of sampling successful individuals. Each generation g, CMA-ES first samples λ individuals $x_1^{(g+1)}, ..., x_\lambda^{(g+1)}$ and chooses the best μ points as the parents of generation *g+1*. Then CMA-ES shifts the mean by a weighted average of μ selected steps:

$$m^{(g+1)} = m^{(g)} + c_m \sum_{i=1}^{\mu} w_i \left(x_{i:\lambda}^{(g+1)} - m^{(g)}\right). \tag{1}$$

where $x_{i:\lambda}$ denotes the i-th best point in terms of the function value and c_m is a learning rate which is usually set to 1. Next, covariance matrix adaptation is performed, which amounts to learning a second order model of the underlying objective function. To control the step-size CMA-ES uses *Cumulative Step Length Adaptation (CSA)* [21]:

$$\sigma^{(g+1)} = \sigma^{(g)} exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||p_\sigma^{(g+1)}||}{E||\mathcal{N}(0,I)||} - 1\right)\right), \tag{2}$$

where $c_\sigma < 1$ is the learning rate, $d_\sigma \approx 1$ is the damping parameter, and $p_\sigma^{(g+1)} \in \mathbb{R}^n$ is the conjugate evolution path at generation *g+1*:

$$p_\sigma^{(g+1)} = (1 - c_\sigma) p_\sigma^{(g)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{eff}} C^{(g)^{-\frac{1}{2}}} \frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}. \tag{3}$$

Note that alternatives for CSA have been proposed, e.g. making use of a success rule [30] or facilitate two-point step-size adaptation [20]. More generally, further research has resulted in many variants of CMA-ES suitable for a variety of different problems. A highly modular framework [48] exists that enables easy choice between 4 608 different versions of CMA-ES. This framework has further been used to demonstrate that, theoretically, switching only once during a run between configurations can yield performance improvements [47]. Simply using the switching rules proposed therein did not yield robust results in practice, but could be improved upon to yield better results [55].

4 Learning Step-Size Adaptation

In this section, we will first discuss how we can model the adaptation of the step-size of CMA-ES as a dynamic algorithm configuration problem and propose to use guided policy search to efficiently find well-performing step-size policies.

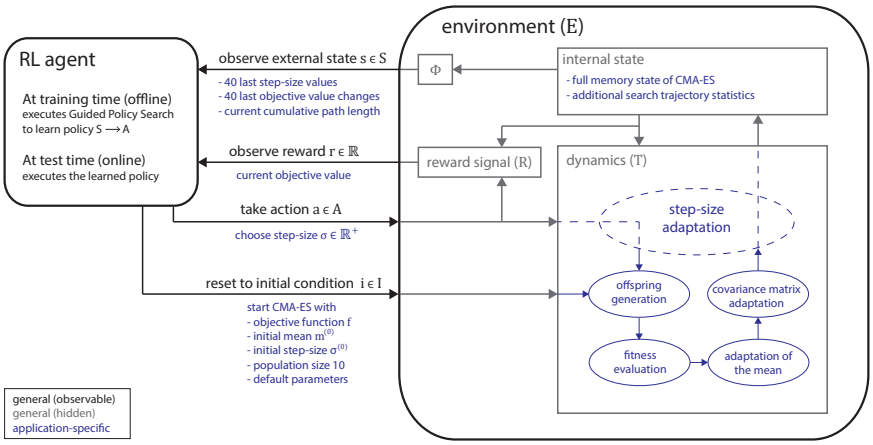


Fig. 1: Interaction of the RL agent with CMA-ES.

4.1 The General Objective

The general objective is to adjust step-size $\sigma^{(g+1)}$ for generation $g+1$ based on some state information s_g on how CMA-ES behaved so far. To achieve that, a probabilistic policy π is responsible for the adjustment:

$$\sigma^{(g+1)} \sim \pi(s_g) \tag{4}$$

Along the lines of DAC, we further say that a policy should not only perform well on a single function f, but generalize to many functions $f \in \mathcal{F}$. Note that the policy must not only depend on features of the search trajectory, but could also be enriched by context information about the function at hand. This allows the policy to easily distinguish between different functions and their characteristics.

Dynamic algorithm configuration allows us to minimize an arbitrary cost function $c : \Pi \times \mathcal{F} \to \mathbb{R}$ that defines how well our algorithm, here CMA-ES, performed by using a policy $\pi \in \Pi$ on a function $f \in \mathcal{F}$. Therefore, our objective is to find a policy π^* that optimally adjusts σ across a set of functions[4]:

$$\pi^* \in \arg\min_{\pi \in \Pi} \sum_{f \in \mathcal{F}} c(\pi, f) \tag{5}$$

4.2 Defining the Components

Having formally described the specific DAC problem at hand, we need to define all the components to apply reinforcement learning (RL) for solving it. In general, the RL paradigm [52] allows learning a policy π mapping observations $\Phi(s') \in S$ of an internal state $s' \in S'$ to actions A by optimizing some reward signal R

[4] We assume that the cost function is well-defined such that an optimal policy exists.

induced by transitions $T : S' \times A \to S'$. So, to solve our DAC problem for step-size adaptation in CMA-ES via RL, we need to define our problem as $\langle S, A, T, R \rangle$, where T is implicitly given by the dynamics of CMA-ES; see Figure 1.

The Step-Size Domain and the Action Space. In principle, the step-size parameter of CMA-ES is a positive, continuous scalar that needs to be adjusted by π. For this, we have two options: (i) discretizing it or (ii) directly optimizing in a continuous domain, which will represent the action space for our RL approach. We argue that the first option is not desirable, as a too fine grid might lead to a large action space with many potentially irrelevant choices; whereas a too coarse grid might not contain all relevant choices. Hence, we model A as the continuous domain of the step size parameter.

State Representation of CMA-ES. To enable DAC for the step-size, it is crucial that S encodes sufficient information about the CMA-ES run. Given that our aim is to learn from, and possibly improve over, the performance of CSA for step-size control, we encode the information CSA uses in the state. Additionally, we include information on the optimization process by keeping a history of a fixed number h of past step-size values (in our experiments, $h = 40$) and past objective values. Specifically, our chosen state description contains:

1. the current step-size value $\sigma^{(g)}$ (see Equation 2)
2. the current cumulative path length $p_\sigma^{(g)}$ (see Equation 2)
3. the history of changes in objective value (i.e. the differences between successive objective values from h previous iterations)
4. the step-size history from h previous iterations[5]

The Cost Function and the Reward. The overall objective of CMA-ES is to find the minimizer of a function f at hand. So, we can say that the cost function should directly reflect the function value found by CMA-ES. Because the optimization budget (e.g., the number of allowed function evaluations) is not always known beforehand, we argue that it is desired to optimize for any-time performance. Since RL maximizes a cumulative reward over time, we can simply define the reward function per step (i.e. generation) as the negative function value of the current incumbent. By doing that, we optimize for any-time performance.

4.3 Using Guided Policy Search for Efficient Learning of the Policy

Prior work showed [49, 51] that value-based RL can be used for learning a DAC policy. However, this approach is typically not very sample-efficient, making it a very expensive approach in general. For example, Biedenkapp et al. [10] needed more than 10 000 algorithm runs to learn a simple sigmoid function.

A key insight of our work is that RL does not need to learn a well-performing policy from scratch but can use existing self-adaptive heuristics as a teacher.

[5] When such a long history is not available yet, the missing values are filled with zeros.

Here, for example, we propose to use CSA as a teacher to learn a policy that either imitates or improves upon it. In addition to better learning stability of the policy, we will show in our experiments that learning a policy for step-size adaptation is comparably cheap by using less than 1 000 runs of CMA-ES.

Similar to Li and Malik [39] in learning to optimize, we propose to use guided policy search (GPS) under unknown dynamics [37] to learn step-size policies. In essence GPS learns arbitrary parameterized policies through supervised learning by fitting a policy to guiding trajectories [38, 37]. From teaching trajectories, a teacher distribution is computed such that it maximizes the reward and the agreement with the current policy. The policy parameters are then updated in a supervised fashion such that new sample trajectories produced by the policy do not deviate too far from the teacher. For a detailed explanation we refer to [37].

4.4 Extending GPS-based DAC by a Stronger Teacher

For our purposes, we initialize the teacher to fit trajectories generated by CMA-ES with CSA. This initial teacher thus closely resembles CSA. As GPS updates the teacher over time to improve the reward, the teacher is likely to move away from CSA over time as only student and teacher are constrained to stay close to each other. If both teacher and student stray too far from CSA, the learned policy might not be able to recover CSAs behaviour in cases where it is beneficial. Thus we would like to encourage the student policy to also continually learn from CSA, to gain a more diverse teaching experience.

Instead of restricting the student policies through hard divergence criterion to not go too far away from CSA, we propose to add additional sample trajectories from running CMA-ES with CSA and not only the teacher to train the student policy. Thereby CSA acts as an additional fixed teacher. We extend GPS by introducing a *sampling rate* to determine the fraction of sample trajectories obtained from CSA when training the policy. Finally, in order to ensure exploration during learning, the initial step-size values and values for the mean of the initial distribution for the functions in the training set are randomly sampled from a uniform distribution.

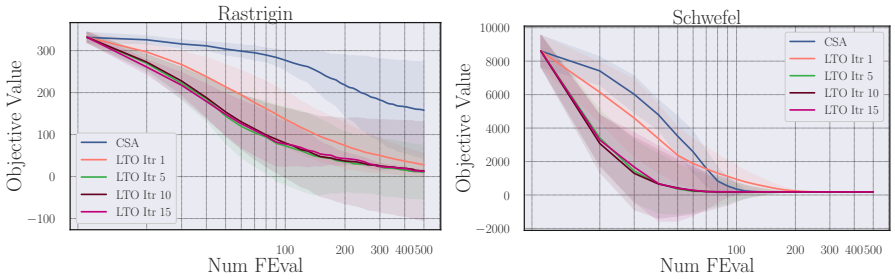
5 Experiments

In this section we empirically evaluate the effectiveness of our proposed approach. We demonstrate the ability of our policies to generalize to new settings.

5.1 Setup

For guided policy search we used the implementation provided by Li and Malik [39]. We incorporated our policy[6] in the python version of CMA-ES (pycma) in version 2.7.0 [22]. We only optimized the step-size adaptation with our approach and left all other pycma parameters as specified by the default, except we

[6] Code and trained policies available at https://github.com/automl/LTO-CMA



(a) (b)

Fig. 2: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our methods incumbent policy after 1, 5, 10 and 15 training iterations on the Rastrigin function (a) and the Schwefel function (b).

used a fixed population size of 10. As functions, we used a representative set with different characteristics as introduced by Hansen et al. in the BBOB-2009 [23].

We used 10 runs of SMAC [28, 40] to tune the initial step-size of CSA for each of the 10 considered BBOB functions individually, giving us a strong baseline. On the unseen functions we used an initial step-size of 0.5. In all experiments we used the same initial step-size for both our method and the baseline.

We trained our step-size policy for 50 steps (i.e. generations) of CMA-ES. We model the policy as a neural network consisting of two hidden layers with 50 hidden units each and ReLU activations. During training, the trajectory samples are obtained from the teaching policy with a probability of 0.7, whereas with a probability of 0.3 we sample trajectories from running CMA-ES with CSA. We obtain the final policy after training for 15 iterations of GPS.

We show performance comparisons of CMA-ES with the learned policy for step-size adaptation and CMA-ES with CSA from 25 independent runs of each method. The tables show an estimate of how likely it is for our learned policy to outperform CSA, based on pairwise comparisons of final objective values from the 25 runs for each method. The online appendix[7] describes this metric in detail, including its relation to statistical significance[8].

5.2 Function-Specific Policy

Comparison against our Teacher CSA We begin by exploring our method's ability to learn step-size policies when trained on a single 10D function for which we sampled 18 different starting points. In each training iteration of GPS, we evaluated CMA-ES 5 times on all starting conditions. In most cases, we already learn a well performing policy after 10 training iterations, which amounts only

[7] https://ml.informatik.uni-freiburg.de/papers/20-PPSN-LTO-CMA.pdf
[8] Estimates $\geq 0.64 \implies$ our learned policy significantly outperformed CSA ($\alpha = 0.05$)

	Sampling Rate									
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
BentCigar	0.53	0.84	0.46	**0.96**	0.38	0.33	0.14	0.26	0.25	0.08
Discus	0.00	0.66	0.23	**0.74**	0.34	0.35	0.30	0.37	0.29	0.32
Ellipsoid	0.59	**0.97**	0.51	**0.97**	0.51	0.48	0.35	0.48	0.56	0.44
Katsuura	0.64	0.91	0.66	**0.96**	0.64	0.63	0.63	0.64	0.64	0.61
Rastrigin	0.81	0.94	0.83	**1.00**	0.97	0.87	0.79	0.85	0.79	0.80
Rosenbrock	0.67	0.28	0.43	**0.89**	0.61	0.17	0.12	0.51	0.57	0.22
Schaffers	0.75	0.68	0.87	0.78	0.92	**0.98**	0.45	0.57	0.90	0.94
Schwefel	0.93	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Sphere	0.77	0.92	0.48	0.78	0.58	0.25	0.94	**0.99**	0.93	0.94
Weierstrass	0.35	**1.00**	0.54	**1.00**	0.32	0.52	0.58	0.52	0.49	0.42
Average	0.60	0.82	0.60	0.91	0.63	0.56	0.53	0.62	0.64	0.58

Table 1: Probability of our method to outperform the baseline when training with different sampling rates. 1.0 indicates that we always outperform the baseline and 0.0 indicates we are always outperformed. The best sampling rate per function are marked in bold.

to $18 \times 5 \times 10 = 900$ runs of CMA-ES. Figure 2 depicts the training performance of our learned step-size policy after 1, 5, 10 and 15 training iterations of GPS on the Rastrigin and Rosenbrock functions. From Figure 2a we can see that even though our policy starts out with samples from the default step-size adaptation of CMA-ES, already after one iteration, the learned policy can outperform the hand-crafted baseline. After four more training steps, our learned policy continues improving and still outperforms CSA. Finally when having trained for 15 iterations, our learned policy readily outperforms CSA, leading not only to a much better final performance, but also to a much better anytime performance on the Rastrigin function. We observe a similar behaviour when training on the Schwefel function, but the learned policy does not drastically outperform CSA.

Studying the Sampling Rate We further used this setting to determine the influence of training length and sampling rate on the final performance of our policies, see Table 1. The sampling rate is crucial for our method as it determines how similar the learned policy's behavior is to CSA.

The performance of the learned policy improved by introducing sample trajectories from CSA compared to only sampling from the time-varying linear Gaussian teacher. Results on some functions are more strongly affected by this change, e.g. BentCigar, than others, such as Schwefel. The final row shows the average performance of the sampling rate over all 10 considered training functions. Further, it becomes apparent that a sampling rate of 0.3 results in the strongest performance of our method, indicating that sampling also from our default teacher can improve performance. As a conclusion of this meta-parameter study, we will use 0.3 for our following experiments on generalization.

	Trajectory Length						
	50	100	150	200	250	500	1000
BentCigar	**0.89**	0.00	0.00	0.00	0.00	0.05	0.04
Discus	0.90	**0.95**	0.76	0.40	0.00	0.00	0.00
Ellipsoid	**0.94**	0.92	0.90	0.86	0.61	0.00	0.00
Katsuura	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rastrigin	**1.00**	0.81	0.80	0.83	0.92	0.73	0.74
Rosenbrock	**0.93**	0.77	0.78	0.90	0.62	0.24	0.04
Schaffers	**0.60**	0.55	0.40	0.39	0.48	0.39	0.57
Schwefel	**0.99**	0.52	0.76	0.79	0.87	0.84	0.65
Sphere	**0.89**	0.00	0.00	0.00	0.00	0.00	0.00
Weierstrass	0.97	0.97	0.89	0.92	**1.00**	**1.00**	**1.00**
Average	0.91	0.65	0.63	0.61	0.55	0.43	0.40

(a) Different Trajectory Lengths

	Dimensions					
	35	40	45	50	55	60
BentCigar	0.87	0.98	0.56	0.49	0.76	**1.00**
Discus	0.89	0.86	0.93	0.94	0.94	**0.97**
Ellipsoid	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Katsuura	0.92	0.92	0.96	**1.00**	0.96	0.87
Rastrigin	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rosenbrock	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Schaffers	0.31	0.58	0.78	**0.87**	0.76	0.74
Schwefel	**1.00**	0.96	0.96	**1.00**	**1.00**	0.98
Sphere	0.41	0.38	0.56	0.65	0.64	**0.72**
Weierstrass	0.97	**1.00**	0.95	**1.00**	**1.00**	0.93
Average	0.84	0.87	0.87	0.89	0.91	0.92

(b) Different # Dimensions

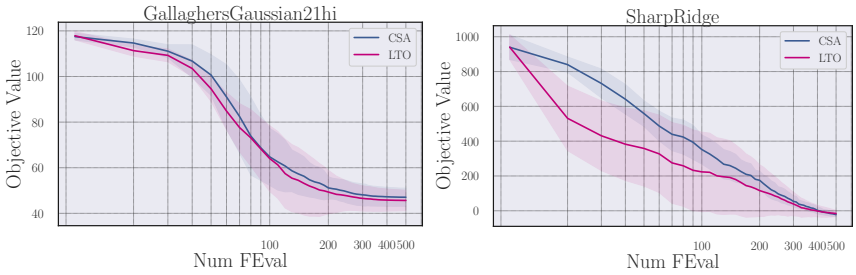
Table 2: Probability of our method to outperform the baseline (a) for varying trajectory lengths, when having only trained with trajectories of length 50, and (b) for different dimensions when training them on functions of dimension $5-30$ and applying the learned policies to functions of dimensionality > 30.

Generalization to Longer Trajectory Length Finally, in this setting we explore the capability of the agent to transfer to longer trajectories. During training we opted to limit the training samples to be of maximal length 50, which corresponds to 500 function evaluations, to keep the cost for training low. Naturally the question thus arises if it is possible to further make use of such policies on longer optimization trajectories. From Table 2a we can observe that, even if a policy is trained with trajectories of at most 500 function evaluations, the policies are generally capable of generalizing to optimization trajectories that are 5 times longer while struggling to generalize to even longer trajectories.[9] On functions where the learned policies perform very well, only a small performance decrease is noticeable over a longer trajectory. On other functions the final performance lacks behind that of the handcrafted baseline over longer optimization trajectories, whereas on Weierstrass, the opposite is the case. On average, we can observe a decline in final performance of our learned policy, the further the optimization trajectory length is from the one used originally for training. A limiting factor as of yet is scaling the training to much longer trajectories. With increased trajectory length more training iterations will be needed to learn well performing policies.

5.3 Function-Class Specific Policy

We are generally not interested in policies that are only of use for one specific function; a more desirable policy would be capable of handling a broader range

[9] The learned policies outperform CSA on anytime performance as shown in the Appendix, but CSA is better in terms of end objective values.



(a) (b)

Fig. 3: Optimization trajectories of CMA-ES using CSA (blue) and our learned policy on two prior unseen test functions. The solid lines depict the mean performance and the shaded area the standard deviation over 25 repetitions.

of functions. As a first step, we are interested in generalizing to similar functions of a specific function class. A very interesting, yet challenging task is hereby to generalize to higher dimensions. For this purpose we trained our policies on functions of dimension $5 - 30$ and evaluated them on dimensions $35 - 60$.

From Table 2b we can see that with increasing dimensionality, the probability that our policies outperform the handcrafted baseline actually *increases*. Upon inspection of the results, we see that with increasing dimensionality, the baseline optimization trajectories need more and more generations before reaching a good performance. Similarly, with increase in dimensionality, optimization trajectories guided by our policies require more generations to reach good final performances, however they are less affected by the dimensionality than the baseline. Especially on functions like Rosenbrock or Ellipsoid this effect seems to be very strongly pronounced. We can observe this trend for both training and testing our policies (see appendix for results on training).

5.4 Generalization to New Functions

Policies scaling to higher dimensions already promise great generalization capability. However, in practice, the problems, to which a solver is applied, could be fairly heterogeneous. To look into a more realistic scenario, we trained our agent on the 10 black-box functions we have mentioned before and assess its generalization capability on 12 black-box functions unseen during training.

Figure 3 shows two exemplary optimization trajectories that are achievable with our learned policies, compared to that of the default CSA. On Gallhager's Gaussian 21-hi we see that the optimization trajectory of CMA-ES with our learned policy closely resembles that of the handcrafted baseline as the step-sizes follow the same trend, see Figure 3a. On SharpRidge (Figure 3b) the learned policy is able to find well performing regions quicker; however in the end the baseline catches up.

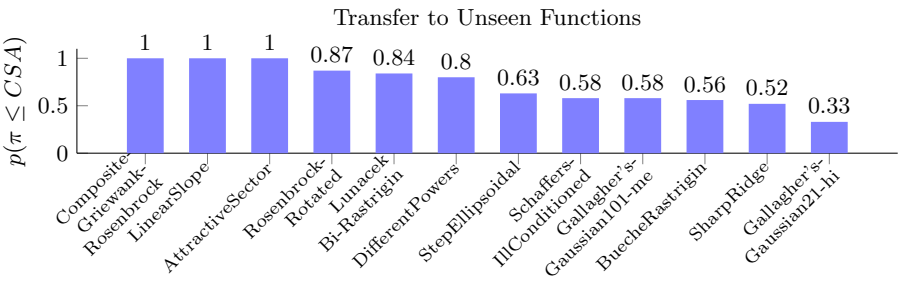


Fig. 4: Probability of our learned policies outperforming the default baseline on prior unseen test functions when training on all 10 BBOB functions.

Figure 4 summarizes the result for all 12 test functions. On 6 out of the 12 test functions, the learned policy significantly (≥ 0.64, $\alpha = 0.05$) outperformed the baseline, while being significantly outperformed (≤ 0.36) on one.

6 Conclusion

We demonstrated that we can automatically learn policies to dynamically configure the mutation step-size parameter of CMA-ES using reinforcement learning. To the best of our knowledge, we are the first to use policy search for the dynamic configuration of evolutionary algorithms, rather than value-based reinforcement learning. In particular, we described how *guided* policy search can be used to learn configuration policies, starting from a known handcrafted default policy. We conducted a comprehensive empirical investigation, and observed that (i) the learned policies are capable of outperforming the default policy on a wide range of black-box optimization problems; (ii) using a fixed teacher can further improve the performance; (iii) our learned policies can generalize to higher dimensions as well as to unseen functions.

These results open the door for promising future research in which policy search is used to learn policies that jointly configure multiple parameters (e.g. population *and* step-size) of CMA-ES. Another line of future research could improve the employed policy search mechanism, e.g. by learning from a variety of teachers at the same time. A more diverse set of teachers, might facilitate even better generalization as the learned policies could make use of strengths of individual teachers on varying problem domains. Finally, the development of a benchmark platform for dynamic algorithm configuration would facilitate apple-to-apple comparisons of different reinforcement learning techniques, driving future research.

Acknowledgements. The authors acknowledge funding by the Robert Bosch GmbH.

Bibliography

[1] Adriaensen, S., Nowé, A.: Towards a white box approach to automated algorithm design. In: Kambhampati, S. (ed.) Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'16). pp. 554–560 (2016)

[2] Aleti, A., Moser, I.: A systematic literature review of adaptive parameter control methods for evolutionary algorithms. ACM Comput. Surv. **49**(3), 56:1–56:35 (2016)

[3] Andersson, M., Bandaru, S., Ng, A.H.: Tuning of multiple parameter sets in evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 533–540 (2016)

[4] Andrychowicz, M., Denil, M., Colmenarejo, S.G., Hoffman, M.W., Pfau, D., Schaul, T., de Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16). pp. 3981–3989 (2016)

[5] Ansótegui, C., Malitsky, Y., Sellmann, M.: Maxsat by improved instance-specific algorithm configuration. In: Brodley, C., Stone, P. (eds.) Proceedings of the Twenty-eighth National Conference on Artificial Intelligence (AAAI'14). pp. 2594–2600. AAAI Press (2014)

[6] Ansótegui, C., Pon, J., Sellmann, M., Tierney, K.: Reactive dialectic search portfolios for maxsat. In: S.Singh, Markovitch, S. (eds.) Proceedings of the Conference on Artificial Intelligence (AAAI'17). AAAI Press (2017)

[7] Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I. (ed.) Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP'09). Lecture Notes in Computer Science, vol. 5732, pp. 142–157. Springer (2009)

[8] Battiti, R., Campigotto, P.: An investigation of reinforcement learning for reactive search optimization. In: Hamadi, Y., Monfroy, E., Saubion, F. (eds.) Autonomous Search, pp. 131–160. Springer (2012)

[9] Battiti, R., Brunato, M., Mascia, F.: Reactive search and intelligent optimization, vol. 45. Springer Science & Business Media (2008)

[10] Biedenkapp, A., Bozkurt, H.F., Eimer, T., Hutter, F., Lindauer, M.: Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In: Lang, J., Giacomo, G.D., Dilkina, B., Milano, M. (eds.) Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20) (Jun 2020)

[11] Cao, Y., Chen, T., Wang, Z., Shen, Y.: Learning to optimize in swarms. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, (NeurIPS'19). pp. 15018–15028 (2019)

[12] Chen, F., Gao, Y., Chen, Z., Chen, S.: Scga: Controlling genetic algorithms with sarsa (0). In: International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06). vol. 1, pp. 1177–1183. IEEE (2005)

[13] Chen, Y., Hoffman, M., Colmenarejo, S., Denil, M., Lillicrap, T., Botvinick, M., de Freitas, N.: Learning to learn without gradient descent by gradient descent. In: Precup, D., Teh, Y. (eds.) Proceedings of the 34th International Conference on Machine Learning (ICML'17). vol. 70, pp. 748–756. Proceedings of Machine Learning Research (2017)

[14] Daniel, C., Taylor, J., Nowozin, S.: Learning step size controllers for robust neural network training. In: Schuurmans, D., Wellman, M. (eds.) Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16). AAAI Press (2016)

[15] Eiben, A., Horváth, M., Kowalczyk, W., Schut, M.: Reinforcement learning for online control of evolutionary algorithms. In: Brueckner, S., Hassas, S., Jelasity, M., Yamins, D. (eds.) Proceedings of Engineering Self-Organising Systems (ESOA). Lecture Notes in Computer Science, vol. 4335, pp. 151–160. Springer (2007)

[16] Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Precup, D., Teh, Y. (eds.) Proceedings of the 34th International Conference on Machine Learning (ICML'17). vol. 70, pp. 1126–1135. Proceedings of Machine Learning Research (2017)

[17] Fuks, L., Awad, N., Hutter, F., Lindauer, M.: An evolution strategy with progressive episode lengths for playing games. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI). pp. 1234–1240. ijcai.org (2019)

[18] Gaspero, L.D., Urli, T.: Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In: Hamadi, Y., Schoenauer, M. (eds.) Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION'12). Lecture Notes in Computer Science, vol. 7219, pp. 384–389. Springer (2012)

[19] Hansen, N.: The CMA evolution strategy: a comparing review. In: Lozano, J., Larranaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a new evolutionary computation. Advances on estimation of distribution algorithms, pp. 75–102. Springer (2006)

[20] Hansen, N.: CMA-ES with two-point step-size adaptation. arXiv:0805.0231 [cs.NE] (2008)

[21] Hansen, N.: The CMA evolution strategy: A tutorial. arXiv:1604.00772v1 [cs.LG] (2016)

[22] Hansen, N., Akimoto, Y., Baudis, P.: CMA-ES/pycma on GitHub. Zenodo, DOI:10.5281/zenodo.2559634 (Feb 2019). https://doi.org/10.5281/zenodo.2559634, https://doi.org/10.5281/zenodo.2559634

[23] Hansen, N., Finck, S., Ros, R., Auger, A.: Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA (2009)
[24] Hansen, N., Ostermeier, A.: Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The $(\mu/\mu_I, \lambda)$-CMA-ES. Proceedings of the 5th European Congress on Intelligent Techniques and Soft Computing p. 650–654 (1997)
[25] Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**, 159–195 (2001)
[26] Hoos, H.: Programming by optimization. Communications of the ACM **55**(2), 70–80 (2012)
[27] Hutter, F., Hoos, H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Lodi, A., Milano, M., Toth, P. (eds.) Proceedings of the Seventh International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR'10). Lecture Notes in Computer Science, vol. 6140, pp. 186–202. Springer (2010)
[28] Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C. (ed.) Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11). Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer (2011)
[29] Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H., Leyton-Brown, K.: The configurable SAT solver challenge (CSSC). Artificial Intelligence **243**, 1–25 (2017)
[30] Igel, C., Hansen, N., Roth, S.: Covariance matrix adaptation for multi-objective optimization. Evolutionary Computation **15**, 1–28 (2001)
[31] Igel, C.: Neuroevolution for reinforcement learning using evolution strategies. In: The 2003 Congress on Evolutionary Computation, 2003. CEC'03. vol. 4, pp. 2588–2595. IEEE (2003)
[32] Kadioglu, S., Sellmann, M., Wagner, M.: Learning a reactive restart strategy to improve stochastic search. In: International Conference on Learning and Intelligent Optimization. pp. 109–123. Springer (2017)
[33] Karafotias, G., Eiben, A., Hoogendoorn, M.: Generic parameter control with reinforcement learning. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. pp. 1319–1326 (2014)
[34] Karafotias, G., Hoogendoorn, M., Eiben, Á.: Parameter control in evolutionary algorithms: Trends and challenges. IEEE Trans. Evolutionary Computation **19**(2), 167–187 (2015)
[35] Karafotias, G., Smit, S., Eiben, A.: A generic approach to parameter control. In: European Conference on the Applications of Evolutionary Computation. pp. 366–375. Springer (2012)
[36] Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the International Conference on Learning Representations (ICLR'15) (2015), published online: iclr.cc
[37] Levine, S., Abbeel, P.: Learning neural network policies with guided policy search under unknown dynamics. In: Ghahramani, Z., Welling, M., Cortes,

C., Lawrence, N., Weinberger, K. (eds.) Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NeurIPS'14). pp. 1071–1079 (2014)
[38] Levine, S., Koltun, V.: Guided policy search. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML'13). pp. 1–9. Omnipress (2013)
[39] Li, K., Malik, J.: Learning to optimize. In: Proceedings of the International Conference on Learning Representations (ICLR'17) (2017), published online: iclr.cc
[40] Lindauer, M., Eggensperger, K., Feurer, M., Falkner, S., Biedenkapp, A., Hutter, F.: SMAC v3: Algorithm configuration in Python. https://github.com/automl/SMAC3 (2017)
[41] Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: Proceedings of the International Conference on Learning Representations (ICLR'19) (2019), published online: iclr.cc
[42] López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. rep., IRIDIA, Université Libre de Bruxelles, Belgium (2011), http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf
[43] López-Ibáñez, M., Stützle, T.: Automatic configuration of multi-objective ACO algorithms. In: Dorigo, M., M-Birattari, Caro, G.D., Doursat, R., Engelbrecht, A.P., Floreano, D., Gambardella, L., Groß, R., Sahin, E., Sayama, H., Stützle, T. (eds.) Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS'10). pp. 95–106. Lecture Notes in Computer Science, Springer (2010)
[44] Maclaurin, D., Duvenaud, D., Adams, R.: Gradient-based hyperparameter optimization through reversible learning. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning (ICML'15). vol. 37, pp. 2113–2122. Omnipress (2015)
[45] Muller, S., Schraudolph, N., Koumoutsakos, P.: Step size adaptation in evolution strategies using reinforcement learning. In: Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600). vol. 1, pp. 151–156. IEEE (2002)
[46] Pettinger, J., Everson, R.: Controlling genetic algorithms with reinforcement learning. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. pp. 692–692 (2002)
[47] van Rijn, S., Doerr, C., Bäck, T.: Towards an adaptive CMA-ES configurator. In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, L.D. (eds.) Proceedings of the 15th International Conference on Parallel Problem Solving from Nature (PPSN'18). Lecture Notes in Computer Science, vol. 11101, pp. 54–65. Springer (2018)
[48] van Rijn, S., Wang, H., van Leeuwen, M., Bäck, T.: Evolving the structure of evolution strategies. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE (2016)
[49] Sakurai, Y., Takada, K., Kawabe, T., Tsuruta, S.: A method to control parameters of evolutionary algorithms by using reinforcement learning. In:

Proceedings of the Sixth International Conference on Signal-Image Technology and Internet Based Systems. pp. 74–79. IEEE (2010)
[50] Salimans, T., Ho, J., Chen, X., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv:1703.03864 [stat.ML] (2017)
[51] Sharma, M., Komninos, A., López-Ibáñez, M., Kazakov, D.: Deep reinforcement learning based parameter control in differential evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 709–717 (2019)
[52] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
[53] Thrun, S., Pratt, L.: Learning to learn. Springer Science & Business Media (2012)
[54] Verdooren, L.: Extended tables of critical values for wilcoxon's test statistic. Biometrika **50**(1-2), 177–186 (1963)
[55] Vermetten, D., van Rijn, S., Bäck, T., Doerr, C.: Online selection of CMA-ES variants. In: Auger, A., Stützle, T. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'19). pp. 951–959. ACM (2019)
[56] Xu, Z., Dai, A.M., Kemp, J., Metz, L.: Learning an adaptive learning rate schedule. arXiv:1909.09712 [cs.LG] (2019)
[57] Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: Proceedings of the International Conference on Learning Representations (ICLR'17) (2017), published online: iclr.cc

CHAPTER 8

Learning Heuristic Selection with Dynamic Algorithm Configuration

The content of this chapter has been published as:

> D. Speck, A. Biedenkapp, F. Hutter, R. Mattmüller, and M. Lindauer (2021). "Learning Heuristic Selection with Dynamic Algorithm Configuration". In: *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21)*. Ed. by H. H. Zhuo, Q. Yang, M. Do, R. Goldman, S. Biundo, and M. Katz. AAAI, pp. 597–605.

Project Idea. The idea was proposed jointly by André Biedenkapp and David Speck following discussions about their respective Ph.D. topics. André Biedenkapp proposed the communication protocol and reward function, whereas David Speck proposed the used state and action spaces. André Biedenkapp proposed to use a DDQN agent, based on the first DAC publication. André Biedenkapp further proposed to generate artificial problem instances to better understand the chosen DDQN agent. David Speck designed the generator for the artificial problem instances and further proposed the theoretical elements of the paper jointly with Robert Mattmüller. Marius Lindauer advised André Biedenkapp on choices of hyperparameters for the RL agents and proposed the analysis of the learned policies.

Implementation and experimentation. Implementation was jointly carried out by André Biedenkapp and David Speck. André Biedenkapp implemented the interface of the algorithms in Python, allowing to use of efficient libraries for reinforcement learning. David Speck implemented the interface in C++, directly as part of the FastDownward system. Both David Speck and André Biedenkapp carried out experiments. André Biedenkapp performed initial experiments which were crucial for the hyperparameter configuration of the RL agent, leading to the final agent design. David Speck carried out the final experiments using the agent setup provided by André Biedenkapp.

Paper writing. An initial draft of the paper was prepared by André Biedenkapp. David Speck improved over this draft and edited it to better fit the ICAPS venue. Robert Mattmüller, Marius Lindauer and Frank Hutter provided valuable feedback on this version of the paper and helped prepare the final version. André Biedenkapp helped with finalizing the paper.

Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

Learning Heuristic Selection with Dynamic Algorithm Configuration

David Speck[1,*], **André Biedenkapp**[1,*], **Frank Hutter**[1,2], **Robert Mattmüller**[1], **Marius Lindauer**[3]
[*]Contact Author, Equal contribution
[1]University of Freiburg, Freiburg, Germany
[2]Bosch Center for Artificial Intelligence, Renningen, Germany
[3]Leibniz University Hannover, Hannover, Germany
⟨speckd, biedenka, fh, mattmuel⟩@cs.uni-freiburg.de, lindauer@tnt.uni-hannover.de

Abstract

A key challenge in satisficing planning is to use multiple heuristics within one heuristic search. An aggregation of multiple heuristic estimates, for example by taking the maximum, has the disadvantage that bad estimates of a single heuristic can negatively affect the whole search. Since the performance of a heuristic varies from instance to instance, approaches such as algorithm selection can be successfully applied. In addition, alternating between multiple heuristics during the search makes it possible to use all heuristics equally and improve performance. However, all these approaches ignore the internal search dynamics of a planning system, which can help to select the most useful heuristics for the current expansion step. We show that dynamic algorithm configuration can be used for dynamic heuristic selection which takes into account the internal search dynamics of a planning system. Furthermore, we prove that this approach generalizes over existing approaches and that it can exponentially improve the performance of the heuristic search. To learn dynamic heuristic selection, we propose an approach based on reinforcement learning and show empirically that domain-wise learned policies, which take the internal search dynamics of a planning system into account, can exceed existing approaches.

Introduction

Heuristic forward search is one of the most popular and successful techniques in classical planning. Although there is a large number of heuristics, it is known that the performance, i.e., the informativeness, of a heuristic varies from instance to instance (Wolpert and Macready 1995). While in optimal planning it is easy to combine multiple admissible heuristic estimates using the maximum, in satisficing planning the estimates of inadmissible heuristics are difficult to combine in general (Röger and Helmert 2010). The reason for this is that highly inaccurate and uninformative estimates of a heuristic can have a negative effect on the entire search process when aggregating all estimates. Therefore, an important task in satisficing planning is to utilize multiple heuristics within one heuristic search.

Röger and Helmert (2010) showed the promise of searching with multiple heuristics, maintaining a set of heuristics, each associated with a separate open list to allow switching between such heuristics. This bypasses the problem of aggregating different heuristic estimates, while the proposed alternating procedure uses each heuristic to the same extent. Another direction is the selection of the best algorithm a priori based on the characteristics of the present planning instance (Cenamor, de la Rosa, and Fernández 2016; Sievers et al. 2019). In other words, different search algorithms and heuristics are part of a portfolio from which one is selected to solve a particular problem instance. This automated process is referred to as algorithm selection (Rice 1976) while optimization of algorithm parameters is referred to as algorithm configuration (Hutter et al. 2009). Both methodologies have been successfully applied to planning (Fawcett et al. 2011, 2014; Seipp et al. 2015; Sievers et al. 2019) and various other areas of artificial intelligence, such as machine learning (Snoek, Larochelle, and Adams 2012) or satisfiability solving (Hutter et al. 2017). However, algorithm selection and configuration ignore the non-stationarity of which configuration performs well. In order to remedy this, Biedenkapp et al. (2020) showed that the problem of selecting and adjusting configurations during the search based on the current solver state and search dynamics can be modelled as a contextual Markov decision process and addressed by standard reinforcement learning methods.

In planning, there is little work that takes into account the search dynamics of a planner to decide which planner to use. Cook and Huber (2016) showed that switching between different heuristic searches (planners) based on the search dynamics obtained during a search leads to better performance than a static selection of a heuristic. However, in this approach, several disjoint searches (planners) are executed, which do not share the search progress (Aine and Likhachev 2016). Ma et al. (2020) showed that a portfolio-based approach that can switch the planner at halftime, depending on the performance of the previously selected one, can improve performance over a simple algorithm selection at the beginning. Recent works have investigated switching between different search strategies depending on the internal search dynamics of a planner (Gomoluch, Alrajeh, and Russo 2019; Gomoluch et al. 2020). One approach that shares the search progress is to maintain multiple heuristics as separate open lists (Röger and Helmert 2010). Furthermore, it has been shown that boosting, i.e., giving preference to heuristics that have recently made progress, can improve

search performance (Richter and Helmert 2009). While in these works heuristic values are computed for each state, Domshlak, Karpas, and Markovitch (2010) investigated the question, whether the time spent for the computation of the heuristic value for a certain state pays off.

Another avenue of work considers how to "directly" create or learn new heuristic functions. One example is the work of Ferber, Helmert, and Hoffmann (2020), which utilizes supervised learning to learn a heuristic function where the input is the planning (world) state itself. Thayer, Dionne, and Ruml (2011) showed that admissible heuristics can be transformed online, into inadmissible heuristics, which makes it possible to tailor a heuristic to a specific planning instance.

In this work, we introduce and define dynamic algorithm configuration (Biedenkapp et al. 2020) for planning, by learning a policy that dynamically selects a heuristic within a search with multiple open lists (Röger and Helmert 2010) based on the current search dynamics. We prove that a dynamic adjustment of heuristic selection during the search can exponentially improve the search performance of a heuristic search compared to a static heuristic selection or a *non-adaptive* policy like alternating. Furthermore, we show that such a dynamic control policy is a strict generalization of other already existing approaches to heuristic selection. We also propose a set of state features describing the current search dynamics and a reward function for training a reinforcement learning agent. Finally, an empirical evaluation shows that it is possible to learn a dynamic control policy on a *per-domain basis* that outperforms approaches that do not involve search dynamics, such as ordinary heuristic search with a single heuristic and alternating between heuristics.

Background

We first introduce classical planning, then discuss greedy best-first search with multiple heuristics, and finally present the concept of dynamic algorithm configuration based on reinforcement learning. Note that the terminology and notation of planning and reinforcement learning are similar, so we use the symbol $\sim$ for all notations directly related to reinforcement learning; e.g. π denotes a plan of a planning task, while $\tilde{\pi}$ is a policy obtained by reinforcement learning.

Classical Planning

A problem instance or task in classical planning, modeled in the SAS$^+$ formalism (Bäckström and Nebel 1995), is a tuple $i = \langle \mathcal{V}, s_0, \mathcal{O}, s_\star \rangle$ consisting of four components. $\mathcal{V}$ is a finite set of state variables, each associated with a finite domain D_v. A fact is a pair (v, d), where $v \in \mathcal{V}$ and $d \in D_v$, and a partial variable assignment over $\mathcal{V}$ is a consistent set of facts, i.e., a set that does not contain two facts for the same variable. If s assigns a value to each $v \in \mathcal{V}$, s is called a state. States and partial variable assignments are functions which map variables to values, i.e., $s(v)$ is the value of variable v in state s (analogous for partial variable assignments). $\mathcal{O}$ is a set of operators, where an operator is a pair $o = \langle pre_o, \mathit{eff}_o \rangle$ of partial variable assignments called preconditions and effects, respectively. Each operator has cost $c_o \in \mathbb{N}_0$. The state s_0 is called the initial state and the partial variable assignment $s_\star$ specifies the goal condition, which defines all possible goal states $S_\star$. With $\mathcal{S}$ we refer to the set of all states defined over $\mathcal{V}$, and with $|i|$ we refer to the size of the planning task i, i.e., the number of operators and facts.

We call an operator $o \in \mathcal{O}$ applicable in state s iff pre_o is satisfied in s, i.e., $s \models pre_o$. Applying operator o in state s results in a state s' where $s'(v) = \mathit{eff}_o(v)$ for all variables $v \in \mathcal{V}$ for which eff_o is defined and $s'(v) = s(v)$ for all other variables. We also write $s[o]$ for s'. The objective of classical planning is to determine a plan, which is defined as follows. A *plan* $\pi = \langle o_0, \ldots, o_{n-1} \rangle$ for planning task i is a sequence of applicable operators which generates a sequence of states $s_0, \ldots, s_n$, where $s_n \in S_\star$ is a goal state and $s_{i+1} = s_i[o_i]$ for all $i = 0, \ldots, n-1$. The cost of plan π is the sum of its operator costs.

Given a planning task, the search for a good plan is called satisficing planning. In practice, heuristic search algorithms such as greedy best-first search have proven to be one of the dominant search strategies for satisficing planning.

Greedy Search with Multiple Heuristics

Greedy best-first search is a pure heuristic search which tries to estimate the distance to a goal state by means of a heuristic function. A heuristic is a function $h : \mathcal{S} \mapsto \mathbb{N}_0 \cup \{\infty\}$, which estimates the cost to reach a goal state from a state $s \in \mathcal{S}$. The perfect heuristic $h^\star$ maps each state s to the cost of the cheapest path from s to any goal state $s_\star \in S_\star$. The idea of greedy best-first search with a single heuristic h is to start with the initial state and to expand the most promising states based on h until a goal state is found (Pearl 1984). During the search, relevant states are stored in an open list that is sorted by the heuristic values of the contained states in ascending order so that the state with the lowest heuristic values, i.e., the most promising state, is at the top. More precisely, in each step a state s with minimal heuristic value is expanded, i.e., its successors $S' = \{s[o] \mid o \in O, s \models pre_o\}$ are generated and states $s' \in S'$ not already expanded are added to the open list according to their heuristic values $h(s')$. Within an open list, for states with the same heuristic value (h-value) the tie-breaking rule that is used is according to the first-in-first-out principle.

In satisficing planning it is possible to combine multiple heuristic values for the same state in arbitrary ways. It has been shown, however, that the combination of several heuristic values into one, e.g. by taking the maximum or a (weighted) sum, does not lead to informative heuristic estimates (Röger and Helmert 2010). This can be explained by the fact that if one or more heuristics provide very inaccurate values, the whole expansion process is affected. Helmert (2006) introduced the idea to maintain multiple heuristics $H = \{h_0, \ldots, h_{n-1}\}$ within one greedy best-first search. More precisely, it is possible to maintain a separate open list for each heuristic $h \in H$ and switch between them at each expansion step while always expanding the most promising state of the currently selected open list. The generated successor states are then evaluated with *each* heuristic and added to the corresponding open lists. This makes it possible to share the search progress (Aine and Likhachev 2016). Especially, an alternation policy, in which all heuris-

tics are selected one after the other in a cycle such that all heuristics are treated and used equally, has proven to be an efficient method (Röger and Helmert 2010). Such equal use of heuristics can help to progress the search space towards a goal state, even if only one heuristic is informative. However, in some cases it is possible to infer that some heuristics are currently, i.e., in the current region of the search space, more informative than others, which is ignored by a strategy like alternation. More precisely, with alternation, the choice of the heuristic depends only on the current time step and not on the current search dynamics or planner state. In general, it is possible to dynamically select a heuristic based on internal information provided by the planner. This is the key idea behind our approach described in the following.

Dynamic Algorithm Configuration

Automated algorithm configuration (AC) has proven a powerful approach to leveraging the full potential of algorithms. Standard AC views the algorithms being optimized as black boxes, thereby ignoring an algorithm's temporal behaviour and ignoring that an optimal configuration might be non-stationary (Arfaee, Zilles, and Holte 2011). Dynamic algorithm configuration (DAC) is a new meta-algorithmic framework that makes it possible to learn to adjust the parameters of an algorithm given a description of the algorithm's behaviour (Biedenkapp et al. 2020).

We first describe DAC on a high level. Given a parameterized algorithm A with its configuration space $\tilde{\Theta}$, a set of problem instances $\mathcal{I}$ the algorithm has to solve, a state description $\tilde{s}_t^i$ of the algorithm A solving an instance $i \in \mathcal{I}$ at step $t \in \mathbb{N}_0$, and a reward signal $\tilde{r}$ assessing the reward (e.g., runtime or number of state expansions) of using a control policy $\tilde{\pi} \in \tilde{\Pi}$ to control A on an instance $i \in \mathcal{I}$, the goal is to find a (*dynamic*) control policy $\tilde{\pi}^* : \mathbb{N}_0 \times \tilde{\mathcal{S}} \times \mathcal{I} \to \tilde{\Theta}$. This policy adaptively chooses a configuration $\tilde{\theta} \in \tilde{\Theta}$ given a state $\tilde{s}_t \in \tilde{\mathcal{S}}$ of A at time $t \in \mathbb{N}_0$ to optimize the reward of A across the set of instances $\mathcal{S}$, i.e., $\tilde{\pi}^* \in \arg\max_{\tilde{\pi} \in \tilde{\Pi}} \mathbb{E}[\tilde{r}(\tilde{\pi}, i)]$. Note that the current time step $t \in \mathbb{N}_0$ and instance $i \in \mathcal{I}$ can be encoded in the state description $\tilde{\mathcal{S}}$ of an algorithm A, which leads to a dynamic control policy, defined as $\tilde{\pi}_{\text{dac}} : \tilde{\mathcal{S}} \to \tilde{\Theta}$.

Figure 1 depicts the interaction between a control policy $\tilde{\pi}$ and a planning system A schematically. At each time step t, the planner sends the current internal state $\tilde{s}_t^i$ and the corresponding reward $\tilde{r}_t^i$ to the control policy $\tilde{\pi}$ based on which the controller decides which parameter setting $h_{t+1} \in \tilde{\Theta}$ to use. The planner progresses according to the decision to the next internal state $\tilde{s}_{t+1}^i$ with reward $\tilde{r}_{t+1}^i$. This formalisation of dynamic algorithm configuration makes it possible to recover prior meta-algorithmic frameworks as special cases which we discuss below.

Dynamic Heuristic Selection

In this section, we will explain how dynamic algorithm configuration can be used in the context of dynamic heuristic selection and how it differs from time-adaptive or in short *adaptive* algorithm configuration and algorithm selection,

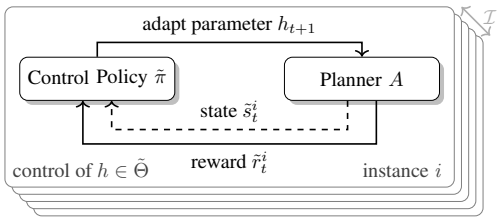


Figure 1: Dynamic configuration of parameter $h \in \tilde{\Theta}$ of algorithm A on an instance $i \in \mathcal{I}$, at time step $t \in \mathbb{N}_0$. Until i is solved or a budget is exhausted, the controller adapts parameter h, based on the internal state $\tilde{s}_t^i$ of A.

which have already been used in the context of search with multiple heuristics. Helmert (2006) introduced the idea of maintaining a set of heuristics H each associated with a separate open list in order to allow the alternation between such heuristics. Considering H as the configuration space $\tilde{\Theta}$ of a heuristic search algorithm A and each state expansion as a time step t, it is possible to classify different dynamic heuristic selection strategies within the framework of dynamic algorithm configuration. For example, alternation is an time-adaptive control policy because it maps each time step to a specific heuristic, i.e., configuration, independent of the instance or the state of the planner. The selection of a particular heuristic depending on the current instance before solving the instance, known as "portfolio planner", is an algorithm selection policy that depends only on the instance and not on the current time step or the internal state of the planner. Exceptions are policies that compare the heuristic values of states, such as the expansion of the state with the overall minimal heuristic value or according to a Pareto-optimality analysis (Röger and Helmert 2010). Such policies depend on the current state of the planner, but ignore the time step and the current instance being solved. This indicates that all three components — instance, time step, and state of the planner — can be important and helpful in selecting the heuristic for the next state expansion. The following summarizes existing approaches to heuristic selection within the framework of algorithm configuration.

- *Algorithm Selection*:
  - Policy: $\tilde{\pi}_{\text{as}} : \mathcal{I} \to H$
  - Example: Portfolios (Seipp et al. 2012; Cenamor, de la Rosa, and Fernández 2016; Sievers et al. 2019)
- *Adaptive Algorithm Configuration*:
  - Policy: $\tilde{\pi}_{\text{aac}} : \mathbb{N}_0 \to H$
  - Example: Alternation (Röger and Helmert 2010; Seipp et al. 2015)
- *Dynamic Algorithm Configuration*:
  - Policy: $\tilde{\pi}_{\text{dac}} : \mathbb{N}_0 \times \tilde{\mathcal{S}} \times \mathcal{I} \to H$
  - Example: Approach proposed in this paper

An Approach based on Reinforcement Learning

In this section, we describe all the parts required to dynamically configure a planning system so that for each individual

time step, a dynamic control policy can decide which heuristic to use based on a dynamic control policy. Here, a time step is a single expansion step of the planning system.

State description. Learning dynamic configuration policies requires descriptive state features that inform the policy about the characteristics and the behavior of the planning system in the search space. Preferably, such features are domain-independent, such that the same features can be used for a wide variety of domains. In addition, such state features should be cheap to compute in order to keep the overhead as low as possible.

As consequence of both desiderata and the intended learning task we propose to use the following state features computed over the entries contained in the corresponding open list of each heuristic:

$\max_h$**:** maximum h value for each heuristic $h \in H$;

$\min_h$**:** minimum h value for each heuristic $h \in H$;

μ_h**:** average h value for each heuristic $h \in H$;

σ_h^2**:** variance of the h values for each heuristic $h \in H$;

$\#_h$**:** number of entries for each heuristic $h \in H$;

t**:** current time/expansion step $t \in \mathbb{N}_0$.

To measure progress, we do not directly use the values of each state feature, but compute the *difference of each state feature* between successive time steps $t-1$ and t. The configuration space is a finite set of n heuristics to choose from, i.e., $\tilde{\Theta} = H = \{h_0, \ldots, h_{n-1}\}$.

The described set of features is a starting point and domain independent, but does not contain any specific context information yet. In general, it is possible to describe an instance or domain with features that describe, for example, the variables, operators or the causal graph (Sievers et al. 2019). If the goal is to learn robust policies that can handle highly heterogeneous sets of instances, it is possible to add contextual information about the planning instance at hand, such as the problem size or the required preprocessing steps (Fawcett et al. 2014), to the state description. However, in this work, we limit ourselves to domain-wise dynamic control policies and show that the concept of DAC can improve heuristic search for this setting in theory and practice.

Reward function. Similar to the state description, the reward function we want to optimize should ideally be domain-independent, cheap and quick to compute. Since the goal is usually to quickly solve as many tasks as possible, a good reward function should reflect this desire.

We use a simple reward of -1 for each expansion step that the planning system has to perform in order to find a solution. Using this reward function, a configuration policy learns to select heuristics that minimize the expected number of state expansions until a solution is found. This reward function ignores aspects such as the quality of a plan, but its purpose is to reduce the search effort and thus improve search performance. Clearly, it is possible to define other reward functions with, e.g., dense rewards to make learning easier. We nevertheless demonstrate that already with our reward function and state features it is possible to learn dynamic control policies, which dominate algorithm selection and adaptive control policies in theory and practice.

Dynamic Algorithm Configuration in Theory

In this section, we investigate the theoretical properties of using DAC for heuristic search algorithms. In optimal planning, where the goal is to find a plan with minimal cost, the performance of heuristic search can be measured by the number of state expansions (Helmert and Röger 2008). This is different for satisficing planning, because plans with different costs can be found and there are generally no "must expand" states that need to be expanded to prove that a solution is optimal. However, the number of state expansions until *any* goal state is found can be used to measure the guidance of a heuristic or heuristic selection (Richter and Helmert 2009; Röger and Helmert 2010).

We want to answer the question of whether it can theoretically be beneficial to use dynamic control policies $\tilde{\pi}_{\text{dac}}$ over algorithm selection policies $\tilde{\pi}_{\text{as}}$ or adaptive algorithm configuration policies $\tilde{\pi}_{aac}$. Proposition 1 proves that for each heuristic search algorithm in combination with each collection of heuristics there is a dynamic control policy $\tilde{\pi}_{\text{dac}}$ which is as good as $\tilde{\pi}_{\text{as}}$ or $\tilde{\pi}_{aac}$ in terms of state expansions.

Proposition 1. *Independent of the heuristic search algorithm and the collection of heuristics, for each algorithm selection policy $\tilde{\pi}_{as}$ and adaptive algorithm configuration policy $\tilde{\pi}_{aac}$ there is a dynamic control policy $\tilde{\pi}_{dac}$ which expands at most as many states as $\tilde{\pi}_{as}$ and $\tilde{\pi}_{aac}$ until a plan is found for a given planning instance.*

Proof. DAC policies generalize algorithm selection and adaptive algorithm configuration policies, thus it is always possible to define $\tilde{\pi}_{\text{dac}}$ as $\tilde{\pi}_{\text{dac}} = \tilde{\pi}_{\text{as}}$ or $\tilde{\pi}_{\text{dac}} = \tilde{\pi}_{\text{aac}}$. □

With Proposition 1 it follows directly that an optimal algorithm configuration policy $\tilde{\pi}^*_{\text{dac}}$ is at least as good as an optimal algorithm selection policy $\tilde{\pi}^*_{\text{as}}$ and an optimal adaptive algorithm configuration policy $\tilde{\pi}^*_{\text{aac}}$:

Corollary 2. *Independent of the heuristic search algorithm and the collection of heuristics, an optimal dynamic control policy $\tilde{\pi}^*_{dac}$ expands at most as many states as an optimal algorithm selection policy $\tilde{\pi}^*_{as}$ and an optimal adaptive algorithm configuration policy $\tilde{\pi}^*_{aac}$ until a plan π is found for a planning task.* □

It is natural to ask to what extent the use of a dynamic control policy instead of an algorithm selection or an adaptive control policy can improve the search performance of heuristic search. We will show that for each algorithm selection policy $\tilde{\pi}_{\text{as}}$ and adaptive algorithm configuration policy $\tilde{\pi}_{\text{aac}}$, we can construct a family of planning tasks so that a dynamic control policy $\tilde{\pi}_{\text{dac}}$ will expand exponentially fewer states until a plan is found. For this purpose, we introduce a family of planning instances i_n with $O(n)$ propositional variables and $O(n)$ operators. The induced transition system of i_n is visualized in Figure 2. There is exactly one goal path s_0, s_1, s_2, which is induced by the unique plan $\pi = \langle o_1, o_2 \rangle$. Furthermore, exactly two states are directly reachable from

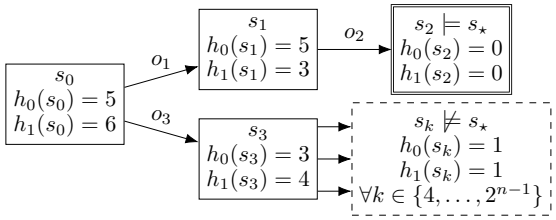


Figure 2: Visualization of the induced transition system of the planning task family i_n.

the initial state, s_1 and s_3. While state s_1 leads to the unique goal state s_2, from s_3 onward exponentially many states $s_4, \ldots, s_{2^n-1}$ in $n = |i_n|$, i.e., $\Omega(2^n) = \Omega(2^{|i_n|})$, can be reached by the subsequent application of multiple actions.

Theorem 3. *For each adaptive algorithm configuration policy $\tilde{\pi}_{aac}$ there exists a family of planning instances i_n, a collection of heuristics H and a dynamic control policy $\tilde{\pi}_{dac}$, so that greedy best-first search with H and $\tilde{\pi}_{aac}$ expands exponentially more states in $|i_n|$ than greedy best-first search with H and $\tilde{\pi}_{dac}$ until a plan π is found.*

Proof. Let $\tilde{\pi}_{\text{aac}}$ be an adaptive algorithm configuration policy. Now, we consider the family of planning tasks i_n (Figure 2) with $|i_n| = O(n)$ and a collection of two heuristics $H = \{h_0, h_1\}$. The heuristic estimates of h_0 and h_1 are shown in Figure 2 and the open lists of greedy best-first search at each time step t are visualized in Figure 3. In time step 0, it is irrelevant which heuristic is selected, always leading to time step 1, where state s_3 is the most promising state according to heuristic h_0, while state s_1 is the most promising state according to heuristic h_1. In time step 1, $\tilde{\pi}_{\text{aac}}$ can either select heuristic h_0 or h_1. We first assume that $\tilde{\pi}_{\text{aac}}$ selects h_0 so that state s_3 is expanded, leading to exponentially many states s_k, which are all evaluated with $h_0(s_k) = h_1(s_k) = 1$ and thus are all expanded before s_1. Therefore, the unique goal state s_2 is found after all other states in the state space $\mathcal{S}$ have been expanded.

In comparison, for $\tilde{\pi}_{\text{dac}}$ we can pick the policy that always selects the heuristic with minimum average heuristic value of all states in the corresponding open list, i.e., $\arg\min_{h \in H} \mu_h$. Following $\tilde{\pi}_{\text{dac}}$, first h_0 and then h_1 is selected, generating the goal state s_2 in time step 1. Therefore, $\tilde{\pi}_{\text{dac}}$ only expands 2 states, while $\tilde{\pi}_{\text{aac}}$ expands 2^{n-2} states until a goal state is found.

Finally, for a policy $\tilde{\pi}_{\text{aac}}$ that selects h_1 at time step 1, it is possible to swap the heuristic estimates of h_0 and h_1 in the constructed collection of heuristics, resulting in the same number of state extensions. □

Theorem 4. *For each algorithm selection policy $\tilde{\pi}_{as}$ there exists a family of planning instances i_n, a collection of heuristics H and a dynamic control policy $\tilde{\pi}_{dac}$, so that greedy best-first search with H and $\tilde{\pi}_{as}$ expands exponentially more states in $|i_n|$ than greedy best-first search with H and $\tilde{\pi}_{dac}$ until a plan π is found.* □

For the proof of this theorem, we refer to the longer arXiv version of this paper (Speck et al. 2020).

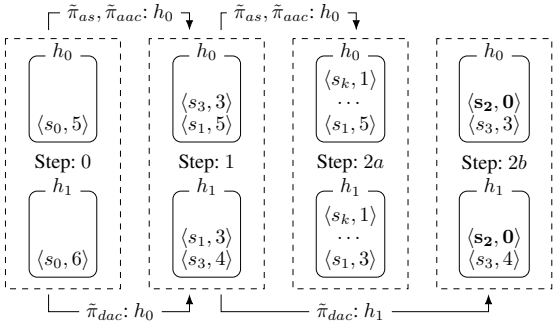


Figure 3: Visualization of two heuristics used to solve an instance of the planning task family i_n.

In Theorems 3 and 4 we assume for simplicity that expanded states are directly removed from all open lists. In practice, open lists are usually implemented as min-heaps, and it is costly to search and remove states immediately. Thus, states that have already been expanded are kept in the open lists and ignored as soon as they have reached the top. We note that this does not affect the theoretical results.

Finally, we want to emphasize that all results presented are theoretical and based on the assumption that it is possible to learn good dynamic control policies. Next, we show that it is possible in practice to learn such dynamic control policies.

Empirical Evaluation

We conduct experiments[1] to measure the performance of our reinforcement learning (RL) approach on domains of the International Planning Competition (IPC). For each domain, the RL policies are trained on a training set and evaluated on a disjoint prior unseen test set of the same domain obtained by a random split. Note that the policies we consider here are not domain-independent, although it is generally possible to add instance- and domain-specific information to the state features. We leave the task of learning domain-independent policies for future work.

Setup

All experiments are conducted with FAST DOWNWARD (Helmert 2006) as the underlying planning system. We use ("eager") greedy best-first search (Richter and Helmert 2009) and min-heaps to represent the open lists (Röger and Helmert 2010). Although there are many more sophisticated search strategies and components, we choose a vanilla search strategy to reduce the factors that might affect the comparison of the actual research question of whether the learned DAC policies can improve the search performance of heuristic search. Nevertheless, our approach is in principle also capable of handling more complex search strategies and components, such as lazy eager search with preferred operators and simple handcrafted DAC like policies such as boosting heuristics (Richter, Westphal, and Helmert 2011).

We implemented an extension for FAST DOWNWARD, which makes it possible to communicate with a controller

[1]Resources: https://github.com/speckdavid/rl-plan

Algorithm	CONTROL POLICY			SINGLE HEURISTIC				BEST AS (ORACLE)		
Domain (# Inst.)	**RL**	RND	ALT	h_{ff}	h_{cg}	h_{cea}	h_{add}	**RL**	ALT	SINGLE h
BARMAN (100)	**84.4**	83.8	83.3	66.0	17.0	18.0	18.0	**89.0**	84.0	67.0
BLOCKSWORLD (100)	**92.9**	83.6	83.7	75.0	60.0	92.0	92.0	**96.3**	88.0	93.0
CHILDSNACK (100)	**88.0**	86.2	86.7	75.0	86.0	86.0	86.0	**88.0**	**88.0**	86.0
ROVERS (100)	95.2	**96.0**	**96.0**	84.0	72.0	68.0	68.0	**96.0**	**96.0**	91.0
SOKOBAN (100)	87.7	87.1	87.0	88.0	**90.0**	60.0	89.0	88.6	87.0	**92.0**
VISITALL (100)	56.9	51.0	51.5	37.0	**60.0**	**60.0**	**60.0**	**61.4**	52.0	60.0
SUM (600)	**505.1**	487.7	488.2	425.0	385.0	384.0	413.0	**519.3**	495.0	489.0

Table 1: Average coverage of different policies for the selection of a heuristic in each expansion step when evaluating the strategies on the prior unseen *test* set. The first three columns are control policies, the next four are individual heuristic searches, while the last three represent the *best algorithm selection* of the corresponding strategies, i.e., oracle selector for each instance.

(dynamic control policy) via TCP/IP and thus to send relevant information (state features and reward) in each time/expansion step and to receive the selected parameter (heuristic). This architecture allows the planner and controller to be decoupled, making it easy to replace components. We considered four different heuristic estimators as configuration space, i.e., $\tilde{\Theta} = H = \{h_{\text{ff}}, h_{\text{cg}}, h_{\text{cea}}, h_{\text{add}}\}$ which can be changed at each time step: (1) the FF heuristic h_{ff} (Hoffmann and Nebel 2001), (2) the causal graph heuristic h_{cg} (Helmert 2004), (3) the context-enhanced additive heuristic h_{cea} (Helmert and Geffner 2008), and (4) the additive heuristic h_{add} (Bonet and Geffner 2001).[2]

For evaluation (final planning runs) we used a maximum of 4 GB memory and 5 minutes runtime. All experiments were run on a compute cluster with nodes equipped with two Intel Xeon Gold 6242 32-core CPUs, 20 MB cache and 188 GB shared RAM running Ubuntu 18.04 LTS 64 bit.

Similar to Biedenkapp et al. (2020), we use ϵ-greedy deep Q-learning in the form of a double DQN (van Hasselt, Guez, and Silver 2016) implemented in CHAINER (Tokui et al. 2019) (CHAINERRL v0.7.0) to learn the dynamic control policies. The networks are trained using ADAM[3] (Kingma and Ba 2014) for 10^6 update steps on a single machine of our cluster with two CPU cores and 20 GB RAM. We use a cutoff of 7 500 control/expansion steps in order to avoid policies being executed arbitrarily long during training. Complex instances may not be solved within this cutoff, even with the optimal policy, and thus learning occurs on simpler instances. However, the underlying assumption is that well performing policies for smaller instances generalize to larger instances within a domain. To determine the quality of a learned policy, we evaluated it every 30 000 steps during training and save the best policy we have seen so far. In total, we performed 5 independent runs of our control policies for each domain, for which we report the average performance. The policies are represented by neural networks for which we determined the hyperparameters in a white-box experiment on a new artificial domain (see Speck et al. (2020)) and kept these hyperparameters fixed for all experiments.

[2]We also conducted additional experiments with five heuristics instead of four, including the $h_{\text{lm-count}}$ heuristic (Richter, Helmert, and Westphal 2008); please see Speck et al. (2020).

[3]We use CHAINER's v0.7.0 default parameters for ADAM.

Experiments

We evaluated the performance of our RL approach on six domains of the International Planning Competition (IPC). These domains were chosen because there are instance generators available online[4] that make it possible to create a suitable number of instances of different sizes. Furthermore, instances of these domains usually require a significant number of state expansions in order to find a plan. For this purpose, we generated 200 instances for all domains and randomly divided them into disjoint training and test sets with the same size of 100 instances each. For each domain we trained five dynamic control policies on the training set and compared them with other approaches on the unseen test set. We are mainly interested in comparing different policies for heuristic selection, which is why, here, the planner always maintains all four open lists, even if only one heuristic is used, and the controller, i.e., the dynamic control policy, alone decides which heuristic is selected.

Table 1 shows the percentage of solved instances per domain, i.e., the average coverage, on the test set. Each domain has a score in the range of 0-100, with larger values indicating more solved instances on average. More precisely, it is possible to obtain a score between 0 and 1 for each planning instance. A value of 0 means that the instance was never solved by the approach, 0.5 means that the instance was solved in half the runs, and 1 means that the instance was always solved. These scores are added up to give the average coverage per domain.

The first three columns correspond to control policies. Entry RL is the average coverage of the five trained dynamic control policies based on reinforcement learning, each averaging over 25 runs with different seeds. Entry RND denotes the average coverage of 25 runs, where a random heuristic is selected in each step. Entry ALT stands for the average over all possible permutations of the execution of alternation. Note that there are $4! = 24$ different ways of executing alternation with four different heuristics. The SINGLE HEURISTIC columns show the coverage when only the corresponding heuristic is used. Finally, the columns for selecting the *best* algorithm selection (BEST AS) stand for the use of an oracle selector, which selects the best configuration of the corresponding technique for each instance. In other

[4]https://github.com/AI-Planning/pddl-generators

words, the best algorithm selection for RL is to choose the best dynamic control policy from the five trained policies for each instance, the best algorithm selection for ALT is to choose the best permutation of alternation for each instance and the best algorithm selection for SINGLE h is to choose the best heuristic for each instance.

Coverage. The results of Table 1 show that our approach (RL) performs best on average in terms of coverage (individual coverage of the five policies: 505.4, 500.6, 501.6, 507.4, 510.1). ALT is slightly better than the uniform randomized choice of a heuristic RND, which indicates that the most important advantage of ALT is to use each heuristic equally with frequent switches and not to switch between them systematically. Furthermore, consistent with the results of Röger and Helmert (2010), single heuristics perform worse than the use of multiple heuristics. Interestingly, in the domain VISITALL, single heuristics have the highest coverage and while RND and ALT have a low coverage, RL performs better. This indicates that in this domain, the dynamic control policies of RL were able to infer that a static policy is well performing or to exclude certain single heuristics. In BLOCKSWORLD, RL has the highest coverage among all approaches. A possible explanation is that a dynamic policy is the key to solving difficult instances in this domain. This assumption is supported by the observation that the best algorithm selection, i.e., the oracle selection of RL, clearly exceeds the other approaches in BLOCKSWORLD. Finally, in ROVER, the use of multiple heuristics seems to be important, and while RL scores better than using single heuristics, the learned policy scores worse than RND and ALT. This may be due to overfitting which we will discuss below. We also compare our approach to the theoretically best possible algorithm selector. Considering the columns BEST AS, we observe that oracle single heuristic selection and oracle alternating selection do not perform better than the average performance of our learned RL policies, which shows that 1) heuristic search with multiple heuristics can in practice benefit from dynamic algorithm configuration and 2) it is possible to learn well performing dynamic policies domain-wise. Even under the unrealistic circumstances of an *optimal* algorithm selector, our learned policies perform better and therefore outperform all possible algorithm selection policies.

If we increase the configuration space by adding another heuristic ($h_{\text{lm-count}}$) the overall coverage of the control policies increases. However, the results are still qualitatively similar to those presented here, showing that the learned DAC policies perform best overall (see Speck et al. (2020)).

We also want to mention the computational overhead of our RL approach compared to ALT and SINGLE HEURISTIC search approaches. While the performance of RL on the test set still exceeds the SINGLE HEURISTIC search of FAST DOWNWARD for all four heuristics with a *single* open list (maintaining only the used heuristic), RL performs slightly worse than the internal heuristic alternation strategy of FAST DOWNWARD. In the future, the overhead can be reduced by integrating the reinforcement learning part directly in FAST DOWNWARD instead of communicating via TCP/IP.

Algorithm	CONTROL POLICY			SINGLE HEURISTIC			
Metric	**RL**	RND	ALT	h_{ff}	h_{cg}	h_{cea}	h_{add}
COVERAGE	**84.2**	81.3	81.4	70.8	64.2	64.0	68.8
GUIDANCE	**38.5**	37.4	37.5	30.8	27.6	28.6	30.4
SPEED	**66.6**	62.8	62.8	54.9	50.4	50.3	54.0
QUALITY	**76.2**	76.0	76.0	65.8	57.6	56.2	60.9

(a) Test set

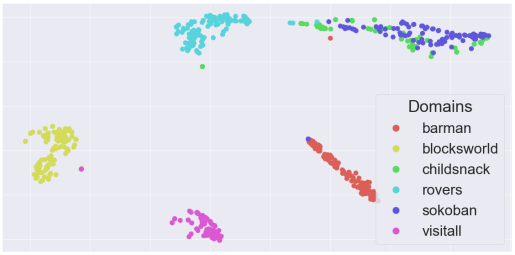
Algorithm	CONTROL POLICY			SINGLE HEURISTIC			
Metric	**RL**	RND	ALT	h_{ff}	h_{cg}	h_{cea}	h_{add}
COVERAGE	**87.0**	83.6	83.0	71.7	64.3	65.0	68.5
GUIDANCE	**39.8**	38.3	38.4	31.4	26.6	28.8	30.2
SPEED	**69.3**	65.3	65.4	56.0	49.1	51.1	54.2
QUALITY	**79.5**	77.9	77.5	66.8	57.3	58.0	61.3

(b) Training set

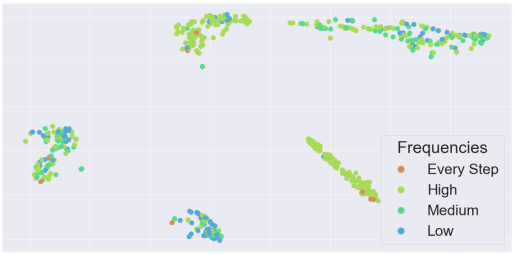
Table 2: A comparison of different control policies and single heuristic search measuring coverage, guidance, speed and solution quality on the prior unseen *test* set (a) and the *training* set (b). Higher scores are preferable for all metrics.

Guidance, speed and quality. Table 2 shows four different metrics including the coverage from above. We additionally evaluate the guidance, speed and quality for each approach with a rating scale (Richter and Helmert 2009; Röger and Helmert 2010). For *guidance*, tasks solved within one state expansion get one point, while unsolved tasks or tasks solved with more than 10^6 state expansions get zero points. Between these extremes the scores are interpolated logarithmically. For *speed* the algorithm gets one point for tasks solved within one second, while the algorithm gets zero points for unsolved tasks or tasks solved in 300 seconds. For *quality* the algorithm gets a score of c^*/c for a solved task, where c is the cost of the reported plan and c^* is the cost of the best plan found with any approach. Finally, the sum of each metric is divided by the number of domains to obtain a total score between 0 and 100. Considering those metrics, control policies perform better than single heuristic approaches. Furthermore, dynamic control polices obtained by RL perform best according to all metrics. However, this analysis favors approaches which solve more instances than others. Recall that plan quality is not taken into account when learning a policy, which explains the small advantage of RL in plan quality, even though more instances have been solved by RL.

Policy analysis. We analyzed the resulting policies on the test set considering all successful runs, i.e., runs where a plan was found. Our first finding is that the RL policies often favor one of the heuristics over all others. In BARMAN, CHILDSNACK, ROVERS, and VISITALL, the h_{ff} heuristic is preferred, while in BLOCKSWORLD the h_{add} and in VISITALL the h_{cea} is most often chosen. Interestingly, the percentage of use of the preferred heuristic varies greatly by domain. In SOKOBAN, e.g., h_{ff} is selected for 98% expansion steps on average, while in BARMAN, the policies only select this heuristic 60% of the time on average.



(a) Policy similarity



(b) Switching frequency

Figure 4: Similarity of the learned RL policies using t-SNE (Hinton and Roweis 2002). Each point represents a run of a policy on a different instance of the test set. The distances between the points represents the similarity of the policies heuristic usage.

This raises the question of how similar the learned policies are and whether the policies are dynamic in the sense that they often switch between different heuristics. To visualize the policies, we used t-distributed stochastic neighbor embedding (t-SNE; Hinton and Roweis 2002), which maps higher-dimensional data into a 2-D space for visualization (Figure 4). As input, we used a vector of four numbers for each instance, indicating the percentage of times a heuristic was chosen, and mapped it to a space where, informally, the more similar the policy executions are, the closer the instances are to each other. Figure 4a shows the similarity of the policy executions for each instance of the different domains. It can be observed that the overall heuristic selection of the learned policies is similar domain-wise, given the resulting clustering. Figure 4b additionally visualizes how often a heuristic was switched, i.e., the frequency of switching the heuristic between subsequent expansion steps. Interestingly, in the majority of the instances, the switching frequency is high (sequences with no h switch < 100 steps), while in the instances corresponding, for example, to the domain SOKOBAN, the switching frequency is low (sequences with no h switch $> 1\,000$ steps). This correlates with the fact that in this domain a single heuristic was selected with a high percentage.

We conclude that our approach is able to learn highly dynamic policies, but also highly static policies, depending on the problem instance at hand. Intuitively, this makes sense, since there may be domains for which a static policy performs best. However, DAC is a generalization of algorithm selection which allows to learn such static policies as well.

Training performance. We compare the performance of our RL approach on the *training* set (Table 2b) with the performance of RL on the test set (Table 2a). Interestingly, RL performs better on the *training* set which can be attributed to a certain degree of overfitting and can explain why in some instances the performance of RL is worse than other approaches on the *test* set. This issue can be addressed by tuning the hyperparameters, expanding the training set or adding further state features. Overall, the improvements DAC yields over the other methods more than outweigh any overfitting, leading to DAC performing best on the test set.

Conclusion

We investigated the use of dynamic algorithm configuration for planning. More specifically, we have shown that dynamic algorithm configuration can be used for dynamic heuristic selection that takes into account the internal search dynamics of a planning system. Dynamic policies for heuristic selection generalize policies of existing approaches like algorithm selection and adaptive algorithm control, and they can improve search performance exponentially. We presented an approach based on dynamic algorithm configuration and showed empirically that it is possible to learn policies capable of outperforming other approaches in terms of coverage.

In future work, we will investigate domain-specific state features to learn domain-independent dynamic policies. Further, it is possible to dynamically control several parameters of a planner and to switch dynamically between different search algorithms. This raises the question how the search progress (Aine and Likhachev 2016) can be shared when using different search strategies. In particular, if we want to combine different search techniques, such as heuristic search (Bonet and Geffner 2001), symbolic search (Torralba et al. 2017; Speck, Geißer, and Mattmüller 2018) and planning as satisfiability (Kautz and Selman 1992; Rintanen 2012), it is an open question how to share the search progress.

Acknowledgments

D. Speck was supported by the German Research Foundation (DFG) as part of the project EPSDAC (MA 7790/1-1). M. Lindauer acknowledges support by the DFG under LI 2801/4-1. A. Biedenkapp, M. Lindauer and F. Hutter acknowledge funding by the Robert Bosch GmbH.

References

Aine, S.; and Likhachev, M. 2016. Search Portfolio with Sharing. In *Proc. ICAPS 2016*, 11–19.

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning Heuristic Functions for Large State Spaces. *AIJ* 175: 2075–2098.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS$^+$ Planning. *Computational Intelligence* 11(4): 625–655.

Biedenkapp, A.; Bozkurt, H. F.; Eimer, T.; Hutter, F.; and Lindauer, M. 2020. Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In *Proc. ECAI 2020*, 427–434.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *AIJ* 129(1): 5–33.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2016. The IBaCoP Planning System: Instance-Based Configured Portfolios. *JAIR* 56: 657–691.

Cook, B.; and Huber, M. 2016. Dynamic heuristic planner selection. In *Proc. SMC 2016*, 2329–2334.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2010. To Max or Not to Max: Online Learning for Speeding Up Optimal Planning. In *Proc. AAAI 2010*, 1071–1076.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. FD-Autotune: Automated Configuration of Fast Downward. In *IPC 2011 planner abstracts*, 31–37.

Fawcett, C.; Vallati, M.; Hutter, F.; Hoffmann, J.; Hoos, H.; and Leyton-Brown, K. 2014. Improved Features for Runtime Prediction of Domain-Independent Planners. In *Proc. ICAPS 2014*, 355–359.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In *Proc. ECAI 2020*, 2346–2353.

Gomoluch, P.; Alrajeh, D.; and Russo, A. 2019. Learning Classical Planning Strategies with Policy Gradient. In *Proc. ICAPS 2019*, 637–645.

Gomoluch, P.; Alrajeh, D.; Russo, A.; and Bucchiarone, A. 2020. Learning Neural Search Policies for Classical Planning. In *Proc. ICAPS 2020*, 522–530.

Helmert, M. 2004. A Planning Heuristic Based on Causal Graph Analysis. In *Proc. ICAPS 2004*, 161–170.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR* 26: 191–246.

Helmert, M.; and Geffner, H. 2008. Unifying the Causal Graph and Additive Heuristics. In *Proc. ICAPS 2008*, 140–147.

Helmert, M.; and Röger, G. 2008. How Good is Almost Perfect? In *Proc. AAAI 2008*, 944–949.

Hinton, G. E.; and Roweis, S. T. 2002. Stochastic Neighbor Embedding. In *Proc. NIPS 2002*, 833–840.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR* 14: 253–302.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *JAIR* 36: 267–306.

Hutter, F.; Lindauer, M.; Balint, A.; Bayless, S.; Hoos, H. H.; and Leyton-Brown, K. 2017. The Configurable SAT Solver Challenge (CSSC). *AIJ* 243: 1–25.

Kautz, H.; and Selman, B. 1992. Planning as Satisfiability. In *Proc. ECAI 1992*, 359–363.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG].

Ma, T.; Ferber, P.; Huo, S.; Chen, J.; and Katz, M. 2020. Online Planner Selection with Graph Neural Networks and Adaptive Scheduling. In *Proc. AAAI 2020*, 5077–5084.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Rice, J. R. 1976. The algorithm selection problem. *Advances in Computers* 15: 65–118.

Richter, S.; and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In *Proc. ICAPS 2009*, 273–280.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *Proc. AAAI 2008*, 975–982.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, 50–54.

Rintanen, J. 2012. Planning as Satisfiability: Heuristics. *AIJ* 193: 45–86.

Röger, G.; and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *Proc. ICAPS 2010*, 246–249.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning Portfolios of Automatically Tuned Planners. In *Proc. ICAPS 2012*, 368–372.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic Configuration of Sequential Planning Portfolios. In *Proc. AAAI 2015*, 3364–3370.

Sievers, S.; Katz, M.; Sohrabi, S.; Samulowitz, H.; and Ferber, P. 2019. Deep Learning for Cost-Optimal Planning: Task-Dependent Planner Selection. In *Proc. AAAI 2019*, 7715–7723.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proc. NIPS 2012*, 2960–2968.

Speck, D.; Biedenkapp, A.; Hutter, F.; Mattmüller, R.; and Lindauer, M. 2020. Learning Heuristic Selection with Dynamic Algorithm Configuration. arXiv:2006.08246 [cs.AI].

Speck, D.; Geißer, F.; and Mattmüller, R. 2018. Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams. In *Proc. ICAPS 2018*, 250–258.

Thayer, J. T.; Dionne, A. J.; and Ruml, W. 2011. Learning Inadmissible Heuristics During Search. In *Proc. ICAPS 2011*, 250–257.

Tokui, S.; Okuta, R.; Akiba, T.; Niitani, Y.; Ogawa, T.; Saito, S.; Suzuki, S.; Uenishi, K.; Vogel, B.; and Vincent, H. Y. 2019. Chainer: A Deep Learning Framework for Accelerating the Research Cycle. In *Proc. KDD 2019*, 2002–2011.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient Symbolic Search for Cost-optimal Planning. *AIJ* 242: 52–79.

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proc. AAAI 2016*, 2094–2100.

Wolpert, D. H.; and Macready, W. G. 1995. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

Part IV

Dynamic Algorithm Configuration: Benchmarking

CHAPTER 9

DACBench: A Benchmark Library for Dynamic Algorithm Configuration

The content of this chapter has been published as:

> T. Eimer, A. Biedenkapp, M. Reimer, S. Adriaensen, F. Hutter, and M. Lindauer (2021b). "DACBench: A Benchmark Library for Dynamic Algorithm Configuration". In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI'21*. Ed. by Z. Zhou. ijcai.org, pp. 1668–1674.

Project Idea. The idea was proposed by Marius Lindauer. Theresa Eimer, André Biedenkapp and Marius Lindauer jointly discussed and decided which potential benchmarks to include in the final collection.

Implementation and experimentation. Implementation was led by Theresa Eimer with support from Maximilian Reimer. André Biedenkapp provided implementations of benchmarks based on earlier publications and advised on the design of DACBench. Steven Adriaensen provided the implementation of the SGD benchmark. Experiments were carried out by Theresa Eimer with support from Maximilian Reimer.

Paper writing. An initial draft of the paper was written by André Biedenkapp and Theresa Eimer with support by Steven Adriaensen. Marius Lindauer and Frank Hutter revised and edited the final version of the paper. The final paper was to a large extent written by André Biedenkapp and Theresa Eimer.

DACBench: A Benchmark Library for Dynamic Algorithm Configuration

Theresa Eimer[1] , **André Biedenkapp**[2] , **Maximilian Reimer**[1] , **Steven Adriaensen**[2] , **Frank Hutter**[2,3] and **Marius Lindauer**[1]
[1]Information Processing Institute (tnt), Leibniz University Hannover, Germany
[2]Department of Computer Science, University of Freiburg, Germany
[3]Bosch Center for Artificial Intelligence, Renningen, Germany
{eimer, reimerm, lindauer}@tnt.uni-hannover.de, {biedenka, adriaens, fh}@cs.uni-freiburg.de

Abstract

Dynamic Algorithm Configuration (DAC) aims to dynamically control a target algorithm's hyperparameters in order to improve its performance. Several theoretical and empirical results have demonstrated the benefits of dynamically controlling hyperparameters in domains like evolutionary computation, AI Planning or deep learning. Replicating these results, as well as studying new methods for DAC, however, is difficult since existing benchmarks are often specialized and incompatible with the same interfaces. To facilitate benchmarking and thus research on DAC, we propose DACBench, a benchmark library that seeks to collect and standardize existing DAC benchmarks from different AI domains, as well as provide a template for new ones. For the design of DACBench, we focused on important desiderata, such as (i) flexibility, (ii) reproducibility, (iii) extensibility and (iv) automatic documentation and visualization. To show the potential, broad applicability and challenges of DAC, we explore how a set of six initial benchmarks compare in several dimensions of difficulty.

1 Introduction

In the last years, algorithm configuration [Ansótegui *et al.*, 2009; Hutter *et al.*, 2011; López-Ibáñez *et al.*, 2016] and in particular automated machine learning [Shahriari *et al.*, 2016; Hutter *et al.*, 2019] offered automatic methods optimizing the settings of hyperparameters to improve the performance of algorithms. However, practitioners of different communities have already known for a while that static hyperparameter settings do not necessarily yield optimal performance compared to dynamic hyperparameter policies [Senior *et al.*, 2013]. One way of formalizing dynamic adaptations of hyperparameters is dynamic algorithm configuration (DAC) [Biedenkapp *et al.*, 2020]. DAC showed its promise by outperforming other algorithm configuration approaches, e.g., choosing variants of CMA-ES [Vermetten *et al.*, 2019] or dynamically adapting its step-size [Shala *et al.*, 2020], dynamically switching between heuristics in AI planning [Speck *et al.*, 2021], or learning learning rate schedules for computer vision [Daniel *et al.*, 2016].

These results, however, also revealed a challenge for the further development of DAC. Compared to static algorithm configuration [Ansótegui *et al.*, 2009; Hutter *et al.*, 2011; López-Ibáñez *et al.*, 2016], applying DAC also requires (i) the definition of a configuration space to search in, (ii) instances to optimize on and (iii) a reward signal defining the quality of hyperparameter settings. However, the optimizer and the algorithm to be optimized have to be integrated much closer in DAC. The current state of the algorithm and the reward function, for example, need to be queried by the optimizer on a regular basis and the applied hyperparameter changes need to be communicated to the algorithm. Therefore, creating reliable, reusable and easy-to-use DAC benchmarks is often fairly hard with no existing standard thus far.

This disparity between benchmarks in addition to the difficulty in creating new ones presents a barrier of entry to the field. Researchers not well versed in both target domain and DAC may not be able to reproduce experiments or understand the way benchmarks are modelled. This makes it hard for pure domain experts to create a DAC benchmark for their domain, severely limiting the number of future benchmarks we can expect to see. A lack of standardized benchmarks, in turn, will slow the progress of DAC going forward as there is no reliable way to compare methods on a diverse set of problems.

To close this gap, we propose DACBench, a suite of standardized benchmarks[1]. On one hand, we integrate a diverse set of AI algorithms from different domains, such as AI planning, deep learning and evolutionary computation. On the other hand, we ensure that all benchmarks can be used with a unified easy-to-use interface, that allows the application of a multitude of different DAC approaches as well as the simple addition of new benchmarks. This paper details the concepts and ideas of DACBench, as well as insights from the benchmarks themselves. Specifically, our contributions are:

1. We propose DACBench, a DAC benchmark suite with a standardized interface and tools to ensure comparability and reproducibility of results;
2. We discuss desiderata of creating DAC benchmarks and how we took them into account in DACBench;
3. We propose a diverse set of DAC benchmarks from different domains showing the breadth of DAC's potential,

[1]The project repository can be found at https://github.com/automl/DACBench

allowing future research to make strong claims with new DAC methods;

4. We show that our DAC benchmarks cover different challenges in DAC application and research.

With this, we strive to lower the barrier of entrance into DAC research and enable research that matters.

2 Related Work

DAC is a general way to formulate the problem of optimizing the performance of an algorithm by dynamically adapting its hyperparameters, subsuming both algorithm configuration (AC) [Hutter *et al.*, 2017] and per-instance algorithm configuration (PIAC) [Ansótegui *et al.*, 2016]. While AC methods can achieve significant improvements over default configurations PIAC algorithms have demonstrated that searching for a configuration per instance can further improve performance. In a similar way, DAC can navigate the over time changing search landscape in addition to instance-specific variations.

Theoretically, this has been shown to be optimal for the $(1 + (\lambda, \lambda))$ genetic algorithm [Doerr and Doerr, 2018], and to enable exponential speedups compared to AC on a family of AI Planning problems [Speck *et al.*, 2021].

Empirically, we have seen dynamic hyperparameter schedules outperform static settings in fields like Evolutionary Computation [Shala *et al.*, 2020], AI Planning [Speck *et al.*, 2021] and Deep Learning [Daniel *et al.*, 2016]. In addition, hyperheuristics [Ochoa *et al.*, 2012] can also be seen as a form of DAC. In this field, it has been shown that dynamic heuristic selection outperforms static approaches on combinatorial optimization problems like Knapsack or Max-Cut [Almutairi *et al.*, 2016].

In the context of machine learning, dynamically adjusting an algorithm's hyperparameters can be seen as a form of learning to learn where the goal is to learn algorithms or algorithm components like loss functions [Houthooft *et al.*, 2018], exploration strategies [Gupta *et al.*, 2018] or completely new algorithms [Andrychowicz *et al.*, 2016]. While DAC does not attempt to replace algorithm components with learned ones, the hyperparameter values of an algorithm are often instrumental in guiding its progress. In some cases they become part of the algorithm. Dynamic step size adaption in ES using heuristics, for example, is very common, but can be replaced and outperformed by more specific DAC hyperparameter policies [Shala *et al.*, 2020].

In other meta-algorithmic areas, reliable and well engineered benchmark libraries also facilitated research progress, incl. ASLib [Bischl *et al.*, 2016], ACLib [Hutter *et al.*, 2014], tabular NAS benchmarks (e.g. [Ying *et al.*, 2019]) and HPOlib [Eggensperger *et al.*, 2013]. In particular, DACBench is strongly inspired by HPOlib and OpenAI gym [Brockman *et al.*, 2016] which also provide a unified interface to benchmarks. Although the hyflex framework [Ochoa *et al.*, 2012] addresses a similar meta-algorithmic problem, in DACBench, we can model more complex problems (i.e., continuous and mixed spaces instead of only categoricals), consider state features of algorithms and cover more AI domains (not only combinatorial problems).

Furthermore DACBench is designed to build upon existing benchmark libraries in target domains by integrating their algorithm implementations. This includes well-established benchmarks like COCO [Hansen *et al.*, 2020] or IOHProfiler [Doerr *et al.*, 2018].

3 Formal Background on DAC

DAC aims at improving a target algorithm's performance through dynamic control of its hyperparameter settings $\lambda \in \Lambda$. To this end, a DAC policy π queries state information $s_t \in \mathcal{S}$ of the target algorithm at each time point t to set a hyperparameter configuration: $\pi : \mathcal{S} \to \Lambda$. Given a starting state s_0 of the target algorithm, a maximal number of solving steps T, a probability distribution p over a space of problem instances $i \in \mathcal{I}$, and a reward function $r_i : \mathcal{S} \times \Lambda \to \mathbb{R}$ depending on the instance i at hand, the objective is to find a policy maximizing the total return:

$$\int_{\mathcal{I}} p(i) \sum_{t=0}^{T} r_i(s_t, \pi(s_t)) \, \mathrm{d}i \tag{1}$$

Following [Biedenkapp *et al.*, 2020], one way of modelling this task is as a contextual MDP $M_{\mathcal{I}} = \{M_i\}_{i \sim \mathcal{I}}$ [Hallak *et al.*, 2015], consisting of $|\mathcal{I}|$ MDPs. Each M_i represents one target problem instance i with $M_i = (\mathcal{S}, \mathcal{A}, \mathcal{T}_i, r_i)$. This formulation assumes that all M_i share a common state space $\mathcal{S}$, describing all possible algorithm states, as well as a single action space $\mathcal{A}$ choosing from all possible hyperparameter configurations Λ. The transition function $\mathcal{T}_i : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, corresponding to algorithm behaviour, and reward function r_i, however, vary between instances.

This formulation allows to apply different configuration approaches on the same problem setting, e.g., algorithm configuration by ignoring all state information ($\pi : \emptyset \to \Lambda$), per-instance algorithm configuration by only taking the instance into account ($\pi : \mathcal{I} \to \Lambda$) or a full DAC agent ($\pi : \mathcal{S} \times \mathcal{I} \to \Lambda$) on the contextual MDP (information about $i \in \mathcal{I}$ is typically directly reflected in $s \in \mathcal{S}$). In view of how close this DAC formulation is to reinforcement learning (RL), in the remainder of the paper we will continue to refer to hyperparameter settings as actions and hyperparameter schedules as policies. Nevertheless, we consider DAC as a general problem that can be solved in different ways, incl. supervised learning, reinforcement learning or even hand-designed policies, e.g., cosine annealing for learning rate adaption in deep learning [Loshchilov and Hutter, 2017] or CSA for CMA-ES [Chotard *et al.*, 2012].

4 DACBench

With DACBench, we strive for an easy-to-use, standardized and reproducible benchmark library that allows evaluating DAC on several, diverse benchmarks. To this end, we will first describe which components are needed to define a DAC benchmark, see Figure 1, and then explain how we can make use of it to ensure our design objectives.

4.1 Components of a DAC Benchmark

Inspired by the flexibility that the modelling as a cMDP allows and the success of OpenAI's gym environments, each

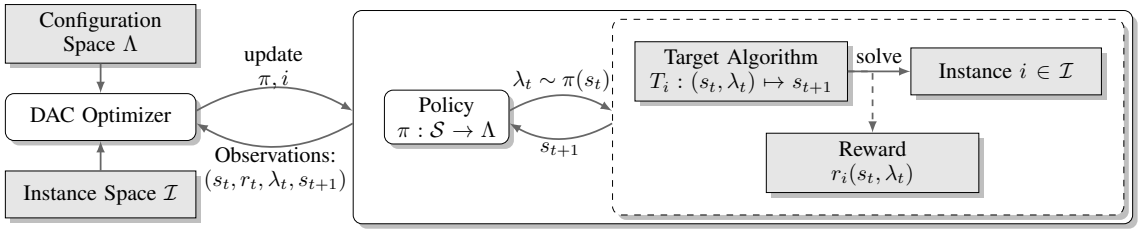


Figure 1: Interaction between optimizer, policy and all components of a DAC benchmark; latter in grey boxes.

DACBench benchmark is modelled along these lines, with the following benchmark-specific design decisions.

Action Space $\mathcal{A}$ describes ways of modifying the current configuration. In the simplest case, the action space directly corresponds to the hyperparameter space, incl. all hyperparameter names and the corresponding ranges.

State Space $\mathcal{S}$ describes available information about the target algorithm state. This can be enriched by context information about the instance at hand. We recommend that it is (i) cheap-to-compute information that is (ii) available at each step and (iii) measures the progress of the algorithm.

Target Algorithm with Transition Dynamics $\mathcal{T}_i$ implicitly defines which states s_{t+1} are observed after hyperparameter configuration λ_t is chosen in state s_t. It is important to fix the target algorithm implementation (and all its dependencies) to ensure that this is reproducible. An implicit design decision of a benchmark here is how long an algorithm should run before the next step description is returned.

Reward Function r_i provides a scalar signal of how well the algorithm can solve a given instance. It is an analogue to the cost function in AC and PIAC and should be the optimization target, e.g., prediction error, runtime or solution quality.

Instance Set $\mathcal{I}$ defines variants of a given problem that has to be solved s.t. the learned policy is able to generalize to new, but similar instances.[2] To assess generalization performance, a training and test set of instances is required. In addition, instances can be described by instance features [Bischl *et al.*, 2016] which facilitates learning of per-instance policies.

This fine granular view on benchmarks allows us on one hand to create a multitude of different benchmarks, potentially with different characteristics. On the other hand, a benchmark in DACBench is a specific instantiated combination of these components s.t. DACBench contributes to reproducible results.

4.2 Practical Considerations & Desiderata

DACBench provides a framework to implement the design decisions above with a focus on accessibility, reproducibility and supporting further research on DAC.

Accessibility So far, applying a new DAC optimizer to a target problem domain requires domain knowledge to be able to interface with a potential algorithm. Comparing optimizers across multiple benchmarks of varying characteristics often requires re-implementing or adapting parts of the optimizers to fit the different interfaces, hurting the consistency of the comparison and taking a lot of time and effort.

Similarly, developing and providing new and interesting benchmarks is challenging as, without a standardized interface, there is little guidance on how to do so. Thus, domain experts wanting to provide a DAC benchmark of a target algorithm often construct their own interface, which can be time-consuming even with a background in MDPs.

Providing a standardized interface would alleviate the issues and facilitate moving DAC as a field forward. Therefore, DACBench provides a common interface for benchmarks, based on OpenAI's gym API [Brockman *et al.*, 2016], that makes interaction with DAC optimizers as simple as possible. This interface is lightweight and intuitive to implement for experts from different domains, encouraging collaboration in the creation of new benchmarks and optimizers. It also allows domain experts to modify existing benchmarks with little effort and minimal knowledge of the base code to create new and interesting variations of known benchmarks, see Appendix C.

Reproducibility As discussed before, adapting an algorithm for DAC can be challenging as there are many design decisions involved. On one hand, to allow studies of new DAC characteristics, we believe it is important to give researchers the flexibility to adjust these components. Therefore, we do not want to propose a framework that fixes too many decision points as it could restrict important future research. On the other hand, we believe there is a need for standardized benchmarks to facilitate comparing different methods as well as reproducing research results. For this purpose, all design decisions of the original experiments should be reproducible. To this end, DACBench includes a mechanism to customize as many of these design decisions as possible, but also to record them such that other researchers can reproduce the experiments (for more details, see Appendix A).

Facilitating Further Research Lastly, DACBench supports researchers by providing resources needed to work on DAC problems as well as thorough documentation of the design decisions of each benchmark. As existing benchmarks are often not documented very well, working with them requires thorough study of the code base. Instead, DACBench provides all important details about individual benchmarks in a concise manner through comprehensive documentation.

Furthermore, DACBench provides quality of life components like structured logging and visualization that make working with DACBench seamless. The logging system gives

[2] For simplicity, we only discuss the case of a set of training instances. In general, DACBench also supports instance generators s.t. the set of instances does not have to be fixed in advance.

users the option to save a variety of details like the policies or state information for later analysis. Further, the built-in visualization tools make evaluating experiments easy (examples include Figures 3, 3 and 5) and can directly use the data provided by the logging system.

These considerations contribute to driving open research on DAC forward by ensuring easy reproducibility of experiments, usability for a diverse audience and sharing of experiment configurations. By adopting a simple yet modular interface, we improve general accessibility to the field as well as the ability to continuously evolve DAC benchmarks.

4.3 Six Initial Diverse DAC Benchmarks

We propose six initial benchmarks for DACBench from different domains and with different challenges (for in-depth descriptions, see Appendix B). We believe these present a versatile set of problems both for testing DAC methods across diverse benchmarks and developing new approaches.

Sigmoid & Luby [Biedenkapp *et al.*, 2020] are time series approximation tasks with no underlying target algorithm. These artificial benchmarks run very quickly, their optimal policies can be computed efficiently for all possible instances (i.e. transformations of the functions themselves) and it is easy to generate instance sets for a wide range of difficulties. Therefore, Sigmoid and Luby are ideal for DAC developers, e.g. to verify that agents can learn the optimal policy or slowly ramp up the instance heterogeneity in order to test its generalization capabilities.

FastDownward [Helmert, 2006] is a state-of-the-art AI Planner, which gives rise to a more complex benchmark. The task here is to select the search heuristic at each step on a specific problem family with two complementary heuristics. This can be considered one of the easier benchmarks even though significant performance gains on competition domains are possible with four commonly used heuristics [Speck *et al.*, 2021]. The basic instance set we provide includes optimal policy information as an upper performance bound.

CMA-ES [Hansen *et al.*, 2003] is an evolutionary strategy, where the DAC task is to adapt the algorithm's steps size [Shala *et al.*, 2020] when solving BBOB functions. However, finding a good solution in this continuous space is potentially harder than the discrete heuristic selection in FastDownward. While optimal policies are unknown for this benchmark, there is a strong established dynamic baseline in CSA [Chotard *et al.*, 2012].

ModEA includes an example of dynamic algorithm selection for variants of CMA-ES on BBOB functions [Vermetten *et al.*, 2019]. In contrast to the CMA-ES benchmark, a combination of 11 EA elements with two to three options each are chosen in each step; this combination makes up the final algorithm. This multi-dimensional, large action space makes the problem very complex. So we expect this to be a hard benchmark, possibly too hard for current DAC approaches to efficiently determine an effective DAC policy.

SGD-DL adapts the learning rate of a small neural network learning a simple image classification task [Daniel *et al.*, 2016]. The small network size allows for efficient development and benchmarking of new DAC approaches. By varying the instance (dataset-seed pairs) and the network architecture, this benchmark nevertheless opens up ample possibility to grow ever harder as DAC advances.

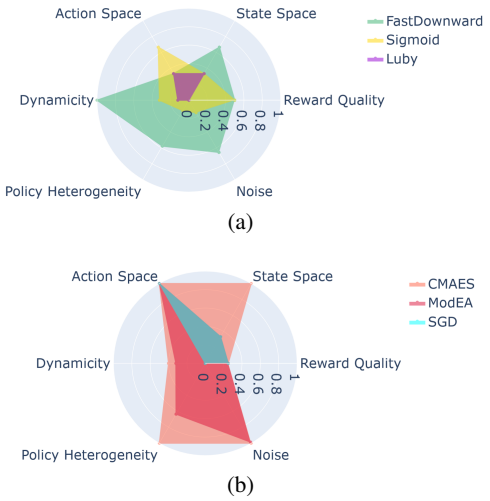


Figure 2: Ranked comparison of difficulty dimensions in DACBench benchmarks. Lower values correspond to easier characteristics.

5 Empirical Insights Gained from DACBench

In order to study our benchmarks, we discuss dimensions of difficulty which are relevant to the DAC setting. To provide insights into how our benchmarks behave in these dimensions, we use static policies, known dynamic baselines and random dynamic policies to explore their unique challenges.

5.1 Setup

To show how our benchmarks behave in practice, we mainly use the static and random policies built into DACBench and, where possible, make use of optimal policies. All of them were run for 10 seeds with at most 1 000 steps on each instance. For benchmarks with a discrete action space, static policies cover all the actions. The two benchmarks with continuous action spaces, CMA-ES and SGD-DL were run with 50 static actions each, distributed uniformly over the action space. For details on the hardware used, refer to Appendix D.

5.2 Coverage of Difficulty Dimensions

Similar to Biedenkapp [2020], we identified six core challenges of learning dynamic configuration policies to characterize our benchmarks. For comparison's sake, we define a scale for each attribute and measure these on our benchmarks. These dimensions of difficulty are: (i) *State* and (ii) *action space size* increase the difficulty of the problem by varying information content, requiring the agent to learn what state information is relevant and which regions in the action space are useful. (iii) *Policy heterogenity* quantifies how successful different policies are across all instances. (iv) *Reward quality* refers to the information content of the given reward signal. (v) *Noise* can disturb the training process through noisy

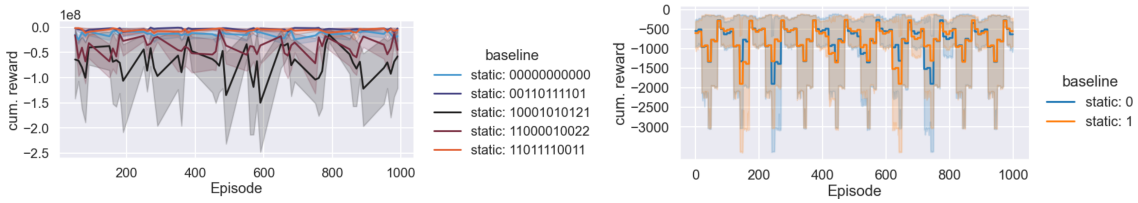


Figure 3: **Left:** Performance of 5 static ModEA policies with 95% confidence interval. The legend shows which components of ModEA were used. **Right:** Comparison of average performance of static FastDownward policies with 95% confidence interval.

transitions or rewards. Lastly, (vi) *dynamicity* shows how frequently the action should be changed, i.e. how complex well-performing policies need to be. See Appendix E for details.

Figure 2 shows how the benchmarks compare with respect to these dimensions of difficulty. While the reward quality is not fully covered, we cover all other dimensions well, with at least a very, moderately and not especially difficult benchmark in each. Additionally, all DACBench benchmarks show a different profile. The data shows that Luby could be considered the easiest of the six, with little noise or policy heterogeneity and a relatively low dynamicity score, requiring only infrequent action changes. SGD-DL's footprint looks similar, though its continuous action space makes for a difficulty spike in that category. While Sigmoid's reward function obscures quite a bit of information, it is not very difficult in the other dimensions. FastDownward on the other hand leads the dynamicity dimension by far, showing a need for more active control. It is also fairly challenging with regard to noise and policy heterogeneity. CMA-ES is even more difficult in these, while also having the largest state space. A more informative reward and lower dynamicity contrast it and other benchmarks. ModEA's difficulty, on the other hand, seems similar except for the challenge of a continuous state space.

While this shows that our benchmark set covers all of our dimensions of difficulty with the exception of reward quality fairly well, we will continue to explore the dimensions of noise, policy heterogeneity and dynamicity in greater detail in order to give a more detailed impression of how these dimensions are expressed.

5.3 Degree of Randomness

To show how randomness is expressed in our benchmarks, we investigate its effects on FastDownward and ModEA.

We quantified randomness by using the standard deviation of the cumulative reward between different seeds for the same actions, each repeated 10 times. ModEA was one of the benchmarks that had a very high relative standard deviation and thus a very high noise score, see Figure 3. While static policies from different parts of the action space vary in performance, their confidence intervals grow much larger the worse they perform. This is to be expected, as policies with a high reward have found EA components that quickly find the optimal solution of the black-box function at hand. If the resulting EA cannot find a solution quickly, the individuals in each generation will have very different proposed solutions, thus resulting in unstable performance. So even though ModEA contains quite a bit of noise, the noise is heteroscedastic, i.e., it is not evenly distributed across the policy space, providing an additional challenge.

FastDownward, on the other hand, also has a high rating in the noise category, but the way its noise is distributed is quite different, see Figure 3. W.r.t. the average performance of both static policies, the 95% confidence interval is up to twice as large as the performance value itself. In contrast to ModEA, the noise is large but likely homoscedastic.

5.4 Effect of Instances

To investigate the effect instances have on our benchmarks, we examine CMA-ES, which showed the highest policy heterogeneity above, and Sigmoid, for which we can compute the optimal policy. CMA-ES and ModEA both operate on instance sets comprised of different function classes between which we can clearly see very different behaviour. The Schaffers function (see Figure 4 left) illustrates that the hand-designed CSA is indeed a good dynamic policy; it outperforms all other static and random policies.

In contrast, CSA performs much worse on the Ellipsoid function (Figure 4 middle). Using the probability estimation proposed by [Shala *et al.*, 2020] based on the Wilcoxon rank sum test, CSA's probability of outperforming any given static policy is 74.6% overall; also shown on a per-instance level in the algorithm footprint [Smith-Miles *et al.*, 2014] in Figure 4. While this shows that CSA's dynamic control policy is preferred on most of CMA-ES instance space, there are also regions that require a different approach, underlining the importance of instance dependent methods.

On the Sigmoid benchmark we see that performance differences between instances persist even for the optimal policy (see Figure 5 left). While it performs very well on some instances, this is far from the case for all of them. Indeed, while it is possible to gain the best possible reward of 10 on some instances, there is an almost even distribution of rewards between the maximum and minimum cumulative reward.

Overall, different instances can have a significant influence on the overall performance, both in terms of which policies are successful on them and how well an agent can do.

5.5 Is Dynamic Better than Static?

Even though we have empirical evidence of DAC agents surpassing static baselines for all of our benchmarks [Daniel *et al.*, 2016; Vermetten *et al.*, 2019; Biedenkapp *et al.*, 2020; Shala *et al.*, 2020; Speck *et al.*, 2021], we analyse and compare the performance of dynamic and static policies on our benchmarks. This way we can estimate the difficulty both in finding a good dynamic policy that surpasses a simple random one but also the difficulty of outperforming the static

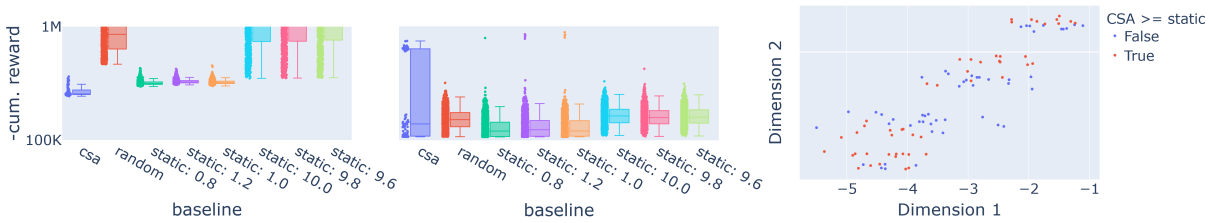


Figure 4: Policy evaluation of CMA benchmark on Schaffers (**left**) and Ellipsoid (**middle**) functions (with 3 best and worst static policies). **Right:** Algorithm footprint t-SNE plot of CMA-ES instances showing where CSA outperforms all static policies.

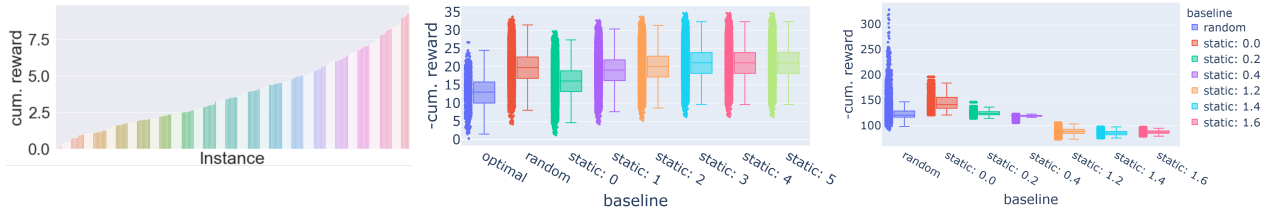


Figure 5: **Left:** Best possible reward for each sigmoid instance. **Middle:** Static and dynamic policies on Luby. The reward is 0 if the agent guesses the correct sequence element, -1 otherwise. **Right:** Static (best and worst 3) and dynamic policies on SGD-DL with $\lambda = 10^{-x}$. The reward here is the validation loss (negative log-likelihood).

policies. Insights into the relationship between static and dynamic policies can highlight characteristics of a benchmark, give upper and lower performance bounds and show the effect size we can expect from DAC approaches in the future.

Our evaluation clearly shows that the benchmarks have a very different sensitivity to dynamic policies. In Luby (Figure 5 middle) we can see that the most common elements of the Luby sequence, elements one and two, outperform the dynamic random policy. As 50% of the Luby sequence consist of the first element and 25% of the second, this is the expected behaviour. Therefore it also makes sense that the optimal policy outperforms all other policies. The random policy does not perform very well, showing that there is a lot of room to improve over it and subsequently over the static policies.

Similarly, the random policy of SGD-DL outperforms some of the worst static policies on average, but does very poorly compared to them on many occasions (see Figure 5 right). Improving over the best static policies here will therefore be much harder for a DAC agent. This is also an example of the fact that dynamically adapting hyperparameters can outperform static settings, as [Daniel *et al.*, 2016] showed for this setting, but the region of well-performing dynamic policies seem to be much smaller than for Luby above. This is the reason for the benchmark's low dynamicity rating. Unlike e.g. FastDownward, which favors frequent action changes regardless of their quality, SGD-DL requires a more subtle approach with more consistency and carefully selected actions.

Therefore, we believe dynamicity will play a large role in how DAC methods should approach benchmarks. While choosing a new action each step for SGD-DL can of course be learned successfully over time, it is a much harder task than Luby. Methods keeping actions for a number of steps at a time may have better success here [Vermetten *et al.*, 2019].

6 Conclusion

We propose DACBench, a standardized benchmark suite for dynamic algorithm configuration (DAC). With it, we provide a framework to configure DAC benchmarks that both enables reproducibility and easy modifications, ensuring that DACBench can help evolve DAC benchmarks further. For example, we plan to extend the FastDownward benchmark beyond single domains and include existing instance features from e.g. Exploratory Landscape Analysis (ELA) for CMA-ES and ModEA. Furthermore, DACBench is easily extendable and we will add new benchmarks, developed by us and the community. As an incentive for researchers to tackle some of the most important difficulties in solving DAC, we provide challenges for several dimensions of hardness. In order to assist in developing these new approaches, we also include tools for tracking important metrics and visualization, making DACBench very easy to use without knowledge of the target domains. Overall, we believe DACBench will make DAC more accessible to interested researchers, make existing DAC approaches more easily comparable and provide a direction for research into new methods. For future work, we plan to build surrogate benchmarks, similar to [Eggensperger *et al.*, 2018] for AC and [Siems *et al.*, 2020] for NAS, to enable DAC benchmarking with minimal computational overhead and minimized CO_2 footprint.

Acknowledgements

We thank Gresa Shala, David Speck and Rishan Senanayake for their contributions to the CMA-ES, FastDownward and SGD-DL benchmarks respectively. Theresa Eimer and Marius Lindauer acknowledge funding by the German Research Foundation (DFG) under LI 2801/4-1. All authors acknowledge funding by the Robert Bosch GmbH.

References

[Almutairi *et al.*, 2016] A. Almutairi, E. Özcan, A. Kheiri, and W. Jackson. Performance of selection hyper-heuristics on the extended hyflex domains. In *Proc. of ISCIS*, 2016.

[Andrychowicz *et al.*, 2016] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *Proc. of NeurIPS*, pages 3981–3989, 2016.

[Ansótegui *et al.*, 2009] C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proc. of CP'09*, pages 142–157, 2009.

[Ansótegui *et al.*, 2016] C. Ansótegui, J. Gabàs, Y. Malitsky, and M. Sellmann. Maxsat by improved instance-specific algorithm configuration. *AIJ*, 235:26–39, 2016.

[Biedenkapp *et al.*, 2020] A. Biedenkapp, H. F. Bozkurt, T. Eimer, F. Hutter, and M. Lindauer. Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In *Proc. of ECAI*, pages 427–434, 2020.

[Bischl *et al.*, 2016] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Frechétte, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren. ASlib: A benchmark library for algorithm selection. *AIJ*, pages 41–58, 2016.

[Brockman *et al.*, 2016] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016.

[Chotard *et al.*, 2012] A. Chotard, A. Auger, and N. Hansen. Cumulative step-size adaptation on linear functions. In *Proc. of PPSN*, 2012.

[Daniel *et al.*, 2016] C. Daniel, J. Taylor, and S. Nowozin. Learning step size controllers for robust neural network training. In *Proc. of AAAI*, 2016.

[Doerr and Doerr, 2018] B. Doerr and C. Doerr. Optimal static and self-adjusting parameter choices for the (1+(λ, λ)) genetic algorithm. *Algorithmica*, 80(5):1658–1709, 2018.

[Doerr *et al.*, 2018] C. Doerr, H. Wang, F. Ye, S. van Rijn, and T. Bäck. Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *arXiv e-prints:1810.05281*, 2018.

[Eggensperger *et al.*, 2013] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13)*, 2013.

[Eggensperger *et al.*, 2018] K. Eggensperger, M. Lindauer, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107(1):15–41, 2018.

[Gupta *et al.*, 2018] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. Meta-reinforcement learning of structured exploration strategies. In *Proc. of NeurIPS*, pages 5307–5316, 2018.

[Hallak *et al.*, 2015] A. Hallak, D. Di Castro, and S. Mannor. Contextual markov decision processes. *CoRR*, abs/1502.02259, 2015.

[Hansen *et al.*, 2003] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computing*, 11(1):1–18, 2003.

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

[Helmert, 2006] M. Helmert. The fast downward planning system. *JAIR*, 26:191–246, 2006.

[Houthooft *et al.*, 2018] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, J. Ho, and Pieter Abbeel. Evolved policy gradients. In *Proc. of NeurIPS*, 2018.

[Hutter *et al.*, 2011] F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION*, pages 507–523, 2011.

[Hutter *et al.*, 2014] F. Hutter, M. López-Ibánez, C. Fawcett, M. Lindauer, H. Hoos, K. Leyton-Brown, and T. Stützle. AClib: a benchmark library for algorithm configuration. In *Proc. of LION*, pages 36–40, 2014.

[Hutter *et al.*, 2017] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. Hoos, and K. Leyton-Brown. The configurable SAT solver challenge (CSSC). *AIJ*, 243:1–25, 2017.

[Hutter *et al.*, 2019] F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2019. Available for free at http://automl.org/book.

[López-Ibáñez *et al.*, 2016] M. López-Ibáñez, J. Dubois-Lacoste, L. Perez Caceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[Loshchilov and Hutter, 2017] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *Proc. of ICLR*, 2017.

[Ochoa *et al.*, 2012] G. Ochoa, M. Hyde, T. Curtois, J. Rodríguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, and E. Burke. Hyflex: A benchmark framework for cross-domain heuristic search. In *Proc. of EvoCOP*, 2012.

[Senior *et al.*, 2013] A. Senior, G. Heigold, M. Ranzato, and K. Yang. An empirical study of learning rates in deep neural networks for speech recognition. In *Proc. of ICASSP*, 2013.

[Shahriari *et al.*, 2016] B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[Shala *et al.*, 2020] G. Shala, A. Biedenkapp, N. Awad, S. Adriaensen, M. Lindauer, and F. Hutter. Learning step-size adaptation in CMA-ES. In *Proc. of PPSN*, pages 691–706, 2020.

[Siems *et al.*, 2020] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter. NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search. *CoRR*, abs/2008.09777, 2020.

[Smith-Miles *et al.*, 2014] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Comput. Oper. Res.*, 45:12–24, 2014.

[Speck *et al.*, 2021] D. Speck, A. Biedenkapp, F. Hutter, R. Mattmüller, and M. Lindauer. Learning heuristic selection with dynamic algorithm configuration. In *Proc. of ICAPS'21*, August 2021.

[Vermetten *et al.*, 2019] D. Vermetten, S. van Rijn, T. Bäck, and C. Doerr. Online selection of CMA-ES variants. In *Proc. of GECCO*. ACM, 2019.

[Ying *et al.*, 2019] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *Proc. of ICML*, 2019.

CHAPTER 10

Theory-inspired Parameter Control Benchmarks for Dynamic Algorithm Configuration

The content of this chapter has been published as:

> A. Biedenkapp, N. Dang, M. S. Krejca, F. Hutter, and C. Doerr (2022a). "Theory-inspired Parameter Control Benchmarks for Dynamic Algorithm Configuration". In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'22)*. Ed. by J. Fieldsend. ACM.

Project Idea. The idea to use this benchmark was proposed by Carola Doerr and Martin Krejca. Martin Krejca proposed the algorithm to compute the optimal policies for a given configuration space. Nguyen Dang and André Biedenkapp proposed to use a DDQN based on previous work.

Implementation and experimentation. Martin Krejca implemented the code and performed all experiments to compute the optimal policies. André Biedenkapp implemented the DDQN agent. Nguyen Dang implemented the benchmark. Nguyen Dang and André Biedenkapp jointly optimized the hyperparameters of the DDQN. Nguyen Dang performed the experiments using the DDQN agent.

Paper writing. Carola Doerr wrote the first draft of the paper. Martin Krejca wrote Section 3. André Biedenkapp contributed to Section 4 and wrote Section 4.1 in full. Nguyen Dang contributed to Section 4 and wrote Sections 4.3 and 4.4 in full. Carola Doerr and Frank Hutter provided feedback and edited this version of the paper. Martin Krejca, Nguyen Dang and André Biedenkapp finalized the paper.

Theory-inspired Parameter Control Benchmarks for Dynamic Algorithm Configuration

André Biedenkapp
University of Freiburg
Freiburg, Germany

Nguyen Dang
University of St Andrews
St Andrews, United Kingdom

Martin S. Krejca
Sorbonne Université, CNRS, LIP6
Paris, France

Frank Hutter
University of Freiburg, Germany
Bosch Center for Artificial Intelligence

Carola Doerr
Sorbonne Université, CNRS, LIP6
Paris, France

ABSTRACT

It has long been observed that the performance of evolutionary algorithms and other randomized search heuristics can benefit from a non-static choice of the parameters that steer their optimization behavior. Mechanisms that identify suitable configurations on the fly ("parameter control") or via a dedicated training process ("dynamic algorithm configuration") are thus an important component of modern evolutionary computation frameworks. Several approaches to address the dynamic parameter setting problem exist, but we barely understand which ones to prefer for which applications. As in classical benchmarking, problem collections with a known ground truth can offer very meaningful insights in this context. Unfortunately, settings with well-understood control policies are very rare.

One of the few exceptions for which we know which parameter settings minimize the expected runtime is the LeadingOnes problem. We extend this benchmark by analyzing optimal control policies that can select the parameters only from a given portfolio of possible values. This also allows us to compute optimal parameter portfolios of a given size. We demonstrate the usefulness of our benchmarks by analyzing the behavior of the DDQN reinforcement learning approach for dynamic algorithm configuration.

CCS CONCEPTS

• **Computing methodologies** → **Randomized search**.

ACM Reference Format:
André Biedenkapp, Nguyen Dang, Martin S. Krejca, Frank Hutter, and Carola Doerr. 2022. Theory-inspired Parameter Control Benchmarks for Dynamic Algorithm Configuration. In *Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3512290.3528846

1 INTRODUCTION

It is well known that the performance of evolutionary algorithms and other black-box optimization heuristics can benefit quite significantly from a non-static choice of the (hyper-)parameters that determine their search behavior [4, 10, 18, 32, 37, 42, 47, 49]. Not only does a dynamic choice of the parameters allow tailoring the search behavior to the problem instance at hand, but it can also be used to leverage complementarity between different search strategies during the different stages of the optimization process, e.g., by moving from a global to a local generation of solution candidates.

Mechanisms to identify suitable dynamic parameter values are intensively studied since decades, see [2, 19, 38] for surveys. Most works focus on generally applicable mechanisms to control the parameters *on-the-fly*, e.g., using self-adaptation [3], success-based parameter update strategies such as the one-fifth success rule [52], co-variance matrix adaptation [32], or reinforcement learning [15] (RL). However, for many practical applications of black-box optimization techniques we also have the possibility to *learn* a parameter control policy via a dedicated training process, either because we anyway need to solve several instances of the same problem or because we can generate instances that are similar to the ones that we expect to see in the future application. Our hope is then to derive structural insight into the algorithms' behavior that can be leveraged to choose their parameters in a more informed manner, just as we are used to do it for classic parameter tuning [8, 34, 58].

The study of parameter control schemes with dedicated offline training is recently enjoying growing attention in the broader AI community, where optimization heuristics are considered an interesting application of AutoML techniques [36]. Examples include the training of a controller for the mutation strategy employed by differential evolution optimizing the CEC2015 problem collection [57] and learning to control the mutation step-size parameter of CMA-ES on the BBOB benchmarks [56]. The problem of training parameter control policies for strong performance on a distribution of instances was coined *dynamic algorithm configuration (DAC)* in [6], where it is formulated as a contextual Markov Decision Process (see Section 4.1 for details). To investigate the functioning and the performance of different DAC approaches, a dedicated library of benchmark problems, *DACBench*, was suggested in [27].

With its rich history of parameter control studies, evolutionary computation has numerous exciting benchmark problems to offer for DAC, e.g., all the problems where dynamic parameter settings have been shown to outperform static ones. One such problem that is particularly well understood is the dynamic fitness-dependent selection of the mutation rates of greedy evolutionary algorithms maximizing the LeadingOnes problem (see Section 2). In particular, we know exactly how the expected runtime of these algorithms depends on the mutation rates used during the run, and this is not

GECCO '22, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9237-2/22/07…$15.00
https://doi.org/10.1145/3512290.3528846

only in asymptotic terms, but also for concrete problem dimensions n [9, 17, 25, 60]. This feature has promoted LeadingOnes as an important benchmark for parameter control studies, both for empirical [21, 25] and for rigorously proven [20, 23, 46] results.

Our in-depth knowledge for LeadingOnes makes the problem an ideal candidate for the in-depth empirical study of the pros and cons of DAC methods: not only does the setting offer relatively fast evaluation times, but we also benefit from a ground truth against which we can compare the policies that are learned during the offline training phase. Existing DAC benchmarks that give access to ground truth either abstract away the actual optimization process and replace it with a simple surrogate or they replace problem instances with unrealistic, artificial proxies. Further, many traditional deep RL benchmarks have deterministic environments, which makes them less representative for the configuration of metaheuristics. LeadingOnes can therefore fill an important gap.

Our Contributions. We demonstrate in this work how the mutation control problem for LeadingOnes can be used to investigate existing DAC approaches and their capabilities. We evaluate a commonly used RL approach using neural networks (dubbed DDQN) and investigate how it scales with different problem dimensions.

Each problem dimension of LeadingOnes provides us with a different problem instance on which we can compare the results of the DAC process to the *ground truth*, i.e., the optimal strategy.[1] To enrich the problem collection further, we also compute optimal control policies for settings in which the algorithms are only allowed to select their parameter values from a given portfolio $\mathcal{K}$ of possible values (Table 2). These results generalize previous works of Lissovoi et al. [46], who analyzed optimal policies for the portfolios that are composed of the integers $i \in [1, k] \cap \mathbb{N}$ for $k \in \Theta(1)$.

We observe for smaller settings, in terms of problem size n and portfolio size k, that the employed DAC method is capable of learning optimal policies quickly (Section 4.3). However, increasing either n or k can drastically increase the learning difficulty, resulting in potentially sub-optimal policies or even no successful learning within the given budget and hyperparameters setting (Figure 8).

Of independent interest for the runtime analysis community are the optimal parameter portfolios (Table 1) that we compute for a number of different combinations of problem dimension n, and portfolio size k. While these optimal portfolios have a large intersection with the `initial_segment` portfolio investigated by Lissovoi et al. [46], the optimal performance achieved with this portfolio is worse than the performance achieved with the portfolio of exponentially growing values $\{2^i \mid i \in [0, k-1] \cap \mathbb{N}\}$.

Outline. In Section 2, we introduce our benchmark, consisting of the LeadingOnes problem as well as the (1+1) RLS algorithm. In Section 3, we explain how to derive optimal policies for a given portfolio. Further, we analyze these policies with respect to increasing portfolio and dimension size. In Section 4, we analyze empirically how well optimal policies can be learned when using the DDQN reinforcement learning approach. Like in Section 3, we consider different portfolios as well as increasing portfolio and dimension sizes. Last, we conclude our work in Section 5.

[1] All optimality claims made here and in the remainder of the paper are always with respect to expected runtime. This is also our primary performance measure, i.e., when we speak of the *performance* of an algorithms, we refer to the expected number of fitness evaluations made before an optimal solution is evaluated for the first time.

Algorithm 1: The (1+1) RLS with state space $\mathcal{S}$, portfolio $\mathcal{K} \subseteq [0..n]$, and parameter selection policy $\pi\colon \mathcal{S} \to \mathcal{K}$, maximizing a function $f\colon \{0,1\}^n \to \mathbb{R}$. See also Section 2.

```
x ← a sample from {0,1}^n chosen uniformly at random;
for t ∈ ℕ do
    s ← current state of the algorithm;
    r ← π(s);
    y ← flip_r(x);
    if f(y) ≥ f(x) then x ← y;
```

Code and Data. Our code and results are on GitHub [7].

2 PARAMETRIZED RLS FOR LEADINGONES

We consider the optimization of the LeadingOnes problem via variants of randomized local search, which we present in the following. We note that we use, for all $a, b \in \mathbb{N}$, the notation $[a..b] := [a, b] \cap \mathbb{N}$.

Parameterized Randomized Local Search. We analyze a parameterized version of the classic randomized local search (RLS) algorithm. While RLS searches only in the direct neighborhood of a current-best solution, its parameterized cousin, the $(1+1)$ RLS (Algorithm 1), can sample solution candidates at larger distances.

The $(1+1)$ RLS maintains a single bit string (the *current solution*), denoted by x in Algorithm 1, initially drawn uniformly at random from $\{0,1\}^n$. Iteratively, the $(1+1)$ RLS generates a new sample y (the *offspring*) from the current solution x, and it replaces x with y if the the objective value $f(y)$ (its *fitness*) is at least as large as $f(x)$. The offspring y is generated by the operator flip_r (the *mutation*), which, given a parameter $r \in [0..n]$, inverts exactly r pairwise different bits in y, chosen uniformly at random from all possible r-subsets of the index set $[1..n]$. We call the parameter r of the mutation the *search radius*. In each iteration, the $(1+1)$ RLS chooses the search radius to apply based on a function π that we call a *(parameter selection) policy*, given some state of the algorithm. The policy π only returns search radii from a certain set $\mathcal{K} \subseteq [0..n]$, which we call the *portfolio* of the algorithm. Note that the portfolio $\mathcal{K}$ and the policy π are part of the input of the $(1+1)$ RLS.

Although information-rich states can prove useful [12], we only have theoretical guarantees for *fitness-dependent* policies, which use exclusively the fitness of the current solution. Doerr and Lengler [24] discuss why it is hard to derive more general bounds. Thus, we assume in this article that the policies are *fitness-dependent*.

Our key performance criterion is the number of iterations until the $(1+1)$ RLS finds a global optimum of its fitness function for the first time, i.e., the smallest $t \in \mathbb{N}$ such that x is optimal at the beginning of that iteration. We refer to this number as the algorithm's *runtime*, noting that it is a random variable.

LeadingOnes. The LeadingOnes problem is defined over bit strings of length $n \in \mathbb{N}$. It asks to maximize the number of leading 1s of a bit string; the all-1s string is the unique global maximum. Formally, $\text{LeadingOnes}\colon \{0,1\}^n \to [0..n], x \mapsto \sum_{i \in [n]} \prod_{j \in [i]} x_j$.

LeadingOnes is a special case of maximizing the longest prefix of agreement with a hidden target bit string $z \in \{0,1\}^n$, evaluated with respect to a hidden permutation σ that shuffles the bit positions, formally defined as $\text{LeadingOnes}_{z,\sigma}\colon \{0,1\}^n \to [0..n], x \mapsto$

$\max\{i \in [0..n] \mid \forall j \in [i] \colon x_{\sigma(j)} = z_{\sigma(j)}\}$. Since the (1 + 1) RLS is unbiased in the sense of Lehre and Witt [45], its performance is identical on each of these problem instances and we thus restrict our attention to the classic LeadingOnes instance mentioned above.

Although $\text{LeadingOnes}_{z,\sigma}$ can be solved using $\Theta(n \log\log n)$ queries in expectation [1], this runtime cannot be achieved with unary unbiased algorithms such as the (1 + 1) RLS. Their runtime grows at least quadratically in the dimension [45]. The same bound of $\Omega(n^2)$ also applies to all (1+1) elitist algorithms [24], of which the (1 + 1) RLS is a representative as well. The expected runtime of the classic RLS with constant search radius 1 is $n^2/2$ [17, Theorem 5].

3 OPTIMAL POLICIES AND PORTFOLIOS FOR LEADINGONES

The exact runtime distribution for LeadingOnes is well understood for the (1 + 1) RLS [17, Section 2.3]. Its expected runtime is, besides its initialization, entirely determined by how quickly it improves the fitness of its current solution. More formally, the most important values are the n different probabilities $(p_i)_{i\in[0..n-1]}$, where, for each $i \in [0..n-1]$, the value p_i denotes the probability that the (1+1) RLS finds a strict improvement if the current solution has fitness i. Choosing for each i the search radius so that p_i is maximized results in an (1 + 1) RLS instance with optimal runtime on LeadingOnes.

In more detail, for each $i \in [0..n-1]$ and each $r \in [0..n]$, let $q(r, i)$ denote the probability that the (1 + 1) RLS finds a strict improvement if the current solution has fitness i and flips exactly r bits during mutation. For LeadingOnes, it holds for all $i \in [0..n-1]$ and all $r \in [0..n]$ that [17, Section 2.3]

$$q(r, i) = \frac{r}{n} \cdot \prod_{j\in[1..r-1]} \frac{n-i-j}{n-j}. \tag{1}$$

An important property of q that allows determining optimal policies for various portfolios of the (1 + 1) RLS is that, for all $i \in [0..n-1]$ and $r \in [0..n-1]$, it holds that [17, Section 2.3]

$$q(r, i) \le q(r+1, i) \text{ if and only if } i \le (n-r)/(r+1). \tag{2}$$

In Section 3.1, we discuss what an optimal policy looks like for the well understood case when permitting *all* possible search radii from 0 to n. We refer to this setting as the *full portfolio*. Afterward, we explain in Section 3.2 how to calculate optimal policies when the portfolio does not contain all search radii, that is, when it is *restricted*. Last, in Section 3.3, we compare optimal policies of different portfolios, including the optimal one, which, given a portfolio size and a problem dimension, minimizes the expected runtime.

Generalizations. Our analyses are easily extended to the (1 + λ) RLS, the variant of the (1 + 1) RLS generating $\lambda \in \mathbb{N}_{\ge 1}$ offspring in each iteration. For it, equation (1) looks slightly different, as it includes λ, but all other arguments work out in the same way.

3.1 Full Portfolio

In the setting of $\mathcal{K} = [0..n]$, an optimal policy π_{opt} satisfies [17, 25]

$$\pi_{\text{opt}} \colon i \mapsto \lfloor n/(i+1) \rfloor \tag{3}$$

as a direct consequence of property (2), as it can be proven that this policy chooses for each fitness i the radius r that maximizes $q(r, i)$.

Note that policy (3) is monotonically decreasing. That is, the higher the fitness of the current individual, the fewer bits are flipped. This entails that not all search radii are used. For example, for a fitness of 0, it is optimal to flip all n bits. For a fitness of 1, it is optimal to flip exactly $\lfloor n/2 \rfloor$ bits. Thus, π_{opt} skips over all search radii in the range $[\lfloor n/2 \rfloor + 1..n-1]$. We further note that using π_{opt} results in an expected runtime of about $0.39n^2$ on LeadingOnes and that using only the search radius 1 results in an expected runtime of $0.5n^2$ [17, Section 2.3]. Thus, the expected runtime of *any* portfolio with search radius 1, using an optimal policy, falls into this range.

3.2 Restricted Portfolio Sizes

For $\mathcal{K} \subsetneq [0..n]$, the optimal policy $\pi_{\text{opt}}^{(\mathcal{K})}$ strongly depends on the search radii in $\mathcal{K}$. Thus, in general, the policy cannot follow an easy formula as given by π_{opt} in policy (3) but needs to be adjusted to the specific values available in $\mathcal{K}$. Further, if $1 \notin \mathcal{K}$, then the expected runtime of an algorithm using $\mathcal{K}$ can be infinite (in particular when the probability of creating a solution with fitness $n-1$ is non-zero, as such a solution can only be improved with search radius 1). Thus, we assume in the following always that $1 \in \mathcal{K}$.

3.2.1 Determining an optimal policy. Let $i \in [0..n-1]$ denote the fitness of the current individual, and assume that $\pi_{\text{opt}}(i) \notin \mathcal{K}$. Due to property (2), q is unimodal in its first component. Thus, the best possible search radius in $\mathcal{K}$ is one of the at most two values closest to $\pi_{\text{opt}}(i)$, i.e., $\pi_{\text{opt}}^{(\mathcal{K})}(i)$ is either $r_i^{\sup} := \max\{r \in \mathcal{K} \mid r < \pi_{\text{opt}}(i)\}$ or $r_i^{\inf} := \min\{r \in \mathcal{K} \mid r > \pi_{\text{opt}}(i)\}$. Thus, it holds that

$$\pi_{\text{opt}}^{(\mathcal{K})}(i) = \arg\max_{r\in\{r_i^{\sup}, r_i^{\inf}\}} q(r, i). \tag{4}$$

Note that this implies that $\pi_{\text{opt}}^{(\mathcal{K})}$ is monotonically decreasing, as, for all $i, j \in [0..n-1]$, $i < j$, it holds that $r_i^{\sup} \ge r_j^{\sup}$ and $r_i^{\inf} \ge r_j^{\inf}$.

Let $\mathcal{D}$ denote the vector of the elements of $\mathcal{K}$ in decreasing order. The monotonicity of equation (4) allows simplifying the calculations for $\pi_{\text{opt}}^{(\mathcal{K})}$ by only determining the fitness values for which the the probability of improvement q for two consecutive elements in $\mathcal{D}$ changes. That is, we only need to determine for all $i \in [1..|\mathcal{K}|-1]$ the largest $j \in [0..n]$ such that $q(\mathcal{D}_i, j) \ge q(\mathcal{D}_{i+1}, j)$. We call each of these $|\mathcal{K}| - 1$ points j a *breaking point*. We note that breaking points do not need to be unique. Algorithm 2 provides a pseudo code for how to determine the breaking points for a given portfolio $\mathcal{K}$. Note that lines 4 to 6 can be improved by applying a binary search that returns the smallest index at which the condition from line 5 holds. This is avoided here in favor of simplicity.

Given the breaking points $(b_i)_{i\in[1..|\mathcal{K}|-1]}$ of a portfolio $\mathcal{K}$ and defining $b_0 = -1$ and $b_{|\mathcal{K}|} = n-1$, the optimal policy $\pi_{\text{opt}}^{(\mathcal{K})}$ is easily calculated by noting that, for all $i \in [0..|\mathcal{K}|]$ and all $j \in [b_i+1..b_{i+1}]$, the i-th largest value in $\mathcal{K}$ is the optimal search radius when the current individual has fitness j.

3.3 Comparing Optimal Policies

We compare different portfolios of the same size k, and we compare their resulting optimal policies calculated as stated at the end of Section 3.2.1. To this end, we consider the following four portfolios. For $n \in \mathbb{N}_{\ge 2}$ and $k \in [2..n]$, we define

- `powers_of_2` to be $\{2^i \mid 2^i \le n \wedge i \in [0..k-1]\}$,
- `initial_segment` to be $[1..k]$,

Algorithm 2: The algorithm to compute, for a given portfolio $\mathcal{K}$ with $1 \in \mathcal{K}$, the breaking points $(b_i)_{i \in [1..|\mathcal{K}|-1]}$ of the optimal policy $\pi_{\text{opt}}^{(\mathcal{K})}$, as discussed in Section 3.2. The function q is defined in equation (1).

$\mathcal{D} \leftarrow \mathcal{K}$ in descending order;
$c \leftarrow 0$;
for $i \in [1..|\mathcal{K}| - 1]$ **do**
  for $j \in [1..n]$ **do**
    if $q(\mathcal{D}_i, j) < q(\mathcal{D}_{i+1}, j)$ **then** break the loop over j;
    $c \leftarrow j$;
  $b_i \leftarrow c$;

Table 1: The optimal portfolios for various sizes k, for problem sizes $n \in \{50, 100\}$, and their expected runtimes (by n^2). For $k = 8$, $n = 100$, computation timed out. See also Section 3.3.

	Optimal portfolio/Expected runtime by n^2			
k	$n = 50$		$n = 100$	
2	1, 4	0.409832	1, 4	0.409897
3	1, 2, 6	0.39568	1, 2, 6	0.395987
4	1, 2, 4, 11	0.3911372	1, 2, 4, 11	0.391403
5	1, 2, 3, 6, 17	0.3895904	1, 2, 3, 6, 16	0.389892
6	1, 2, 3, 5, 9, 21	0.3888308	1, 2, 3, 5, 9, 23	0.389109
7	1, 2, 3, 4, 6, 12, 29	0.388452	1, 2, 3, 4, 6, 11, 27	0.3887584
8	1, 2, 3, 4, 6, 9, 19, 50	0.3882052	–	–

- `evenly_spread` to be $\{i \cdot \lfloor n/k \rfloor + 1 \mid i \in [0..k-1]\}$, and
- `optimal`, which we determine by a brute-force approach over all k-subsets of n that contain the search radius 1. The portfolio with the lowest expected runtime among all of these subsets is considered `optimal`.

Note that `powers_of_2` is only defined for values k of at most $\lfloor \log_2 n \rfloor$. For any larger value of k, it is not defined. Last, note that although there is only one `optimal` *portfolio*, all *policies* discussed in this section are optimal with respect to their specified portfolio.

The portfolio `optimal`. Table 1 shows optimal portfolios for $n \in \{50, 100\}$ and for $k \in [2..8]$. For these cases, the portfolio consists of the interval $[1..\lceil k/2 \rceil]$ and of some larger values that seem to grow exponentially. That is, `optimal` is a mixture of `initial_segment` and a variant of `powers_of_2`. Interestingly, for $k = 8$, the portfolio contains the search radius $50 = n$, which is only relevant if the current individual has a fitness of 0. Due to the uniform initialization, we see this value with 50 %, and we transition to a different state with probability 1 by flipping all bits, so that the difference between the optimal expected runtime that can be achieved with a portfolio of size $k = 8$ over that for $k = 7$ is at most 0.5. Further, `optimal` is identical for $n \in \{50, 100\}$ for $k \in \{2, 3, 4\}$. For larger k, some larger search radii change slightly. This suggests that the generals range of optimal search radii to use is only slightly affected by the problem size.

Optimal policies. Table 2 shows optimal policies (depicted as their relative breaking points) for different portfolio sizes k and problem dimensions n. For `powers_of_2` and `initial_segment`, when increasing k, the portfolio is extended by adding larger search radii. This is reflected in their respective (optimal) portfolio, as the breaking points are also extended. In contrast, for `evenly_spread`, a portfolio of one size is *not* an extension of one of a smaller size. This is reflected in the breaking points, which are not extended for increasing k. For all cases of n and k depicted, `powers_of_2` and `initial_segment` share at least half of their breaking points with `optimal`. This follows also from the results of Table 1, which shows that the high overlap of `optimal` with `initial_segment` continues, whereas the one with `powers_of_2` is not that prominent for larger k. Since all portfolios except for `evenly_spread` contain at least the search radii 1 and 2, the optimal policies also utilize the full range of these radii, following policy (3). For `evenly_spread`, mostly the search radius 1 is important.

Table 2: The breaking points (Algorithm 2) of different portfolios (Section 3.3) of size $k \in \{3, 4\}$ for $n \in \{50, 100\}$. Each breaking point is divided by n. Recall that the breaking points refer to the portfolio sorted in descending order.

k	Portfolio	$n = 50$	$n = 100$
3	`optimal`	0.22, 0.48	0.23, 0.49
	`powers_of_2`	0.26, 0.48	0.28, 0.49
	`initial_segment`	0.3, 0.48	0.32, 0.49
	`evenly_spread`	0, 0.12	0, 0.08
4	`optimal`	0.1, 0.26, 0.48	0.12, 0.28, 0.49
	`powers_of_2`	0.14, 0.26, 0.48	0.15, 0.28, 0.49
	`initial_segment`	0.22, 0.3, 0.48	0.24, 0.32, 0.49
	`evenly_spread`	0, 0.02, 0.16	0, 0.01, 0.1

Figure 1 investigates the case of $k = 3$ for $n = 50$ more closely. We computed for all $\binom{50}{2}$ portfolios of size 3 that contain the search radius 1 the expected runtime of an optimal policy. The figure depicts cumulative data of these computations. Interestingly, the curve follows an almost linear trend, except for the last 5 %, where the increase in the expected runtime is diminishing. This suggests that choosing portfolios uniformly at random has a fair chance of resulting in a good expected runtime of its optimal policy.

In Figure 2, we take a closer look at the impact of the portfolio size k on the expected runtime. It compares the expected runtimes of all four different portfolios when using an optimal policy. Interestingly, although `initial_segment` shares a large part of its search radii with `optimal` (Table 1), the expected runtime of `powers_of_2` is better than that of `initial_segment`. This suggests that having *some* larger search radii is more beneficial than covering only small search radii. However, the comparably bad expected runtime of `evenly_spread` shows that having more than a single small search radius (e.g., 1 *and* 2) drastically improves the expected runtime.

4 ALGORITHM CONFIGURATION WITH REINFORCEMENT LEARNING

Parameter control with a dedicated offline training phase has long been studied [see e.g., 10, 39, 40, 57, 63]. Recently it gained attention in the broader AI community where *dynamic algorithm configuration* (DAC) [6] was proposed as a generalization over algorithm configuration [35] and algorithm selection [53]. In DAC, reinforcement

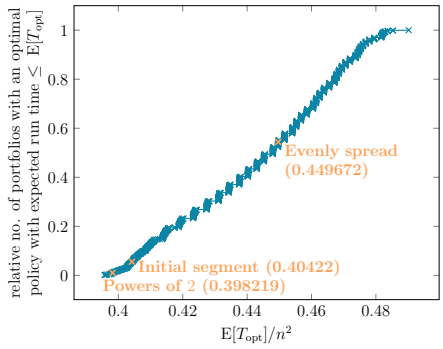


Figure 1: The cumulative fraction of how many out of *all* portfolios have at most the expected (relative) runtimes stated by the x-axis, for $n = 50$. All portfolios have cardinality exactly 3 and contain the search radius 1. Their expected runtime is determined by applying an optimal policy. See also Section 3.3.

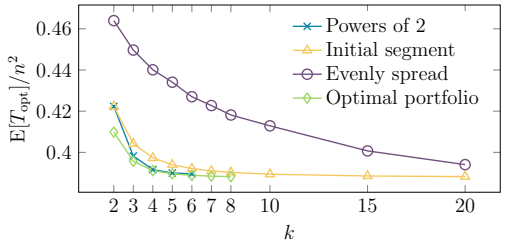


Figure 2: The expected runtimes for the optimal policies of the stated portfolios for $n = 50$. The runtime is divided by n^2. See also Section 3.3. Note that `powers_of_2` is not defined for $k > 6$. Further, we only computed `optimal` up to $k = 8$.

learning (RL) is predominantly used to learn dynamic configuration policies. In the DAC setting, our proposed benchmark is of particular interest as it readily allows us to investigate important questions such as: i) Can DAC learn optimal policies? ii) How does the choice of elements of the portfolio $\mathcal{K}$ influence the learning procedure? iii) How does the size of $\mathcal{K}$ influence the learning procedure? iv) How does the problem size influence the learning procedure?

We recap the most important definitions for DAC in Section 4.1. The experimental setup of our work is summarized in Section 4.2. Results for small portfolios $|\mathcal{K}| \in \{3, 4, 5\}$ and for fixed dimension $n = 50$ are presented in Section 4.3 and results for broader ranges of portfolio sizes and dimensions are discussed in Section 4.4.

4.1 The DAC Framework

The process of dynamically adapting hyperparameters is modeled as a contextual Markov Decision Process (cMDP) [31]. An MDP $\mathcal{M}$ is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and reward function $\mathcal{R}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The transition function describes the dynamics of the process and gives the probability of reaching a state s' when playing action a in state s. Similarly, the reward function describes the reward obtained by playing action a in s. Depending on the system an MDP describes, the reward function can be stochastic. A cMDP extends this formalism through the use of so-called *context information* $i \sim \mathcal{I}$. The context influences the behavior of the reward and transition functions but leaves the state and action spaces unchanged. Thus a cMDP $\mathcal{M} = \{\mathcal{M}_i\}_{i\sim\mathcal{I}}$ is a collection of MDPs with shared state and action spaces, but with individual transition and reward functions. In DAC, the state space describes the internal behavior of an algorithm A (e.g., internal statistics of A) when running it on an instance i (i.e., the context) and the action space is given by the possible values of parameters of A. In practice, the transition and reward functions are unknown and not trivial to approximate or learn. Still, there exist solution approaches for MDPs that do not need direct access to these.

Reinforcement learning (RL) [61] has been demonstrated to be able to learn dynamic configuration policies directly from data [see e.g., 5, 6, 15, 16, 43, 44, 51, 54, 57]. In an offline learning phase, an RL agent interacts with its environment (i.e., the algorithm that is being configured) to learn which actions lead to the highest reward over multiple episodes (trajectory until a goal state or a maximal step-limit is reached). In a trial-and-error fashion, an RL agent iteratively observes the current state s_t of the environment at time t. Based on this observation it selects an action a_t which advances the environment to the next state s_{t+1} and produces a reward signal r_{t+1}. This information is sufficient to learn the value of each state and how to select the next action to maximize the expected reward.

In the commonly used Q-learning approach [64] the goal is to learn the Q-function $Q\colon \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ that maps a state–action pair to the cumulative future reward that is received after playing an action a in state s. The Q-function can be learned in a typical error correction fashion. Given a state s_t and action a_t, the Q-value $Q(s_t, a_t)$ can be updated using temporal differences (TD) as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big(\overbrace{\big(r_t + \gamma \max Q(s_{t+1}, \cdot)\big)}^{\text{TD-target}} - Q(s_t, a_t)\Big)}_{\text{TD-delta}},$$

where α is the *learning rate* and γ is the *discounting factor*. The TD-target is the reward r_t incurred by playing a_t in s_t together with the discounted maximal future reward. The discounting factor determines how important future rewards are when updating the Q-function. The TD-delta then describes how correct or wrong the prediction was and is used to update the Q-function accordingly. The learning rate determines the strength with which the TD-delta updates the original prediction. A reward-maximizing policy can then be defined by only using the learned Q-function as $\pi(s) = \arg\max_{a\in\mathcal{A}} Q(s, \cdot)$. For better exploration while learning, typically ϵ-greedy exploration is used, where ϵ gives the probability that an action a_t is replaced with a randomly sampled one.

Mnih et al. [48] proposed to model the Q-function as a neural network (referred to as deep Q-network) and showed that this allowed to learn Q-functions even for high-dimensional states such as frames of video games. van Hasselt et al. [62] showed that using a single network when selecting the maximizing action in the TD-target and in the prediction of the value often leads to instabilities

due to overestimation during training. To mitigate this, they proposed to use a second copy of the weights of the neural network. One set is used to select the maximizing action and the other is used to predict the value. The second set of weights is kept frozen for short periods at a time and then copied over from the first set for increased stability of predictions. This extension is dubbed double deep Q-network (DDQN) and generally results in overall faster learning due to less overestimation. DDQN has been used as solution approach to DAC problems in DE [57] and AI planning [59].

4.2 Experimental Setup

Following Biedenkapp et al. [6], in our experiments we use a small DDQN with two hidden layers and 50 units each to learn the Q-function. The action space $\mathcal{A}$ is the portfolio $\mathcal{K}$. We define $s_t = f(x_t)$ and $r_t = f(x_t) - f(x_{t-1}) - 1$, where x_t is the solution found by the $(1+1)$ RLS at time step t. During the training of DDQN, we impose a cutoff time of $0.8n^2$ steps per episode to avoid wasting too much time sampling with bad policies. Recall that the expected run time of the simple setting with a constant policy $\pi\colon s \mapsto 1$ is $0.5n^2$ [17]. The episode-cutoff time for our RL training is chosen such that policies slightly worse than this trivial constant policy can still be explored during the learning phase. All DDQN agents are trained with a batch size of 2048, an ϵ-greedy value of 0.2, and a discount factor γ of 0.9998. The batch size determines how many samples are used to compute the gradients when updating the neural network. A larger batch size results in a more accurate estimation of the gradient but takes longer to compute.

It is known that hyperparameters play a crucial role in deep RL algorithms [33]. Tuning them is expensive and not trivial and many purpose-built methods exist depending on the target application [50]. It is, however, not well understood how the hyperparameters influence the learning behavior of agents, especially outside of the domain of video game playing. We built our choice of hyperparameters on prior literature using RL for dynamic tuning and adjusted batch size and γ based on results of a small prestudy (see [7]).

4.3 Results for $n = 50$

In the first set of experiments, we consider a fixed problem size of $n = 50$ as well as the three portfolio settings `initial_segment`, `powers_of_2`, and `evenly_spread` from Section 3.3. For each setting, three portfolio sizes $k \in \{3, 4, 5\}$ are considered. The aim is to study the impact of portfolio settings and portfolio sizes on DDQN's learning behaviors. For each pair of portfolio settings and sizes, a DDQN agent is trained with a budget of 1 million time steps and a walltime limit of 24 hours on an 8-core Intel Xeon E5-4650L computer (2.6 GHz). The best policy is chosen at the end of the training phase and is then evaluated and compared against the optimal policy of the same portfolio $\mathcal{K}$ via 2000 runs (per policy).

As shown in Figure 3, the performance of the DDQN policies is highly comparable to the optimal ones. DDQN is able to reach the performance of the optimal policy within 100 000 time steps in all cases. The learned policies are also quite similar to the optimal ones, with some slight discrepancy, as illustrated in Figure 4, where DDQN learned policies for two example settings (`evenly_spread` with $k = 3$, and `powers_of_2` with $k = 5$).

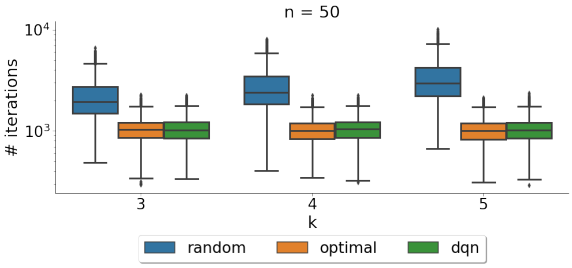


Figure 3: Performance of DDQN and optimal policies on three portfolio settings and three portfolio sizes ($n = 50$).

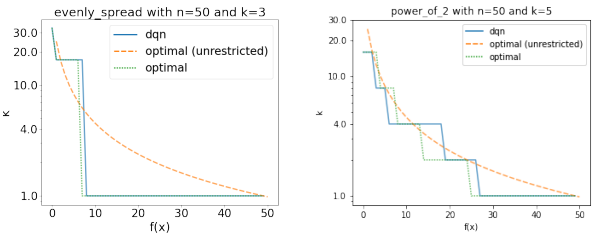


Figure 4: Two example DDQN best learned policies vs. the optimal policy for the same portfolio, and the optimal policy with unrestricted portfolio.

We now have a closer look at the training progress of each RL agent to see how different portfolio settings and portfolio sizes impact the learning behavior of DDQN. To this end, we evaluate the learned policy during each DDQN training at every 2000 time steps via 50 runs and compare it with the optimal policy. Figure 5 shows two example training progress plots of `evenly_spread` and `initial_segment`. Although DDQN frequently reaches the optimal area in both settings, there is a clear distinction between them: for `evenly_spread`, DDQN's performance constantly jumps up and down with very high variance, while for `initial_segment`, the performance progress is much smoother. To quantify these properties of the training progress, we define two metrics for each DDQN training run: (i) *hitting ratio* – the frequency of evaluations in which the expected optimal performance is reached within 0.25 % of its standard deviation; and (ii) *ruggedness* – the standard deviation of performance difference between every pair of consecutively evaluated policies. As shown in Figure 6, the RL agent gets the highest hitting ratios with `evenly_spread`, followed by `powers_of_2` and `initial_segment`. This can be explained due to the actions for `evenly_spread` being very different from each other, some of which often perform very badly in general. Such differences can result in strong signals received by the agent during the training for distinguishing between good and bad policies, which can then help speed up the learning but also causes the landscapes to be less smooth (i.e., high ruggedness) due to the large variance of performance between different policies. Similarly, `initial_segment` has the smallest difference between actions, and the RL agent has the lowest hitting ratios but smoother learning progress.

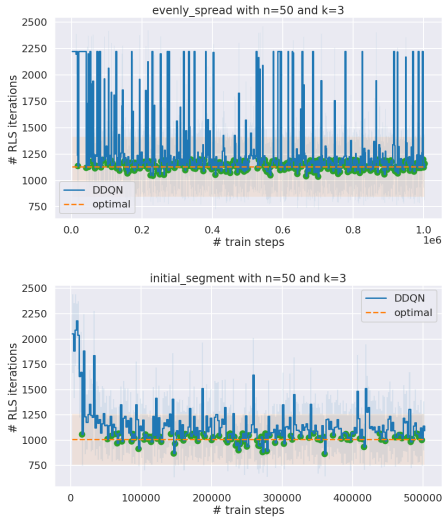


Figure 5: DDQN progress on `evenly_spread` and `initial_segment`. At the green dots, the learned policies reach 0.25 % standard deviation of the optimal policy's performance.

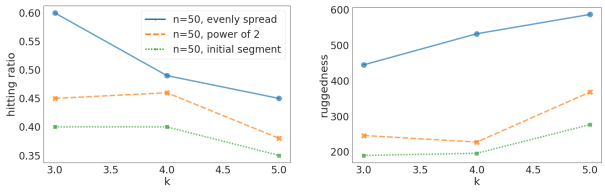


Figure 6: Hitting ratios and ruggedness of DDQN training progress for three portfolio settings ($n = 50$).

4.4 Analyzing the Impact of Portfolio Size and Problem Dimension

Figure 6 indicates a strong relation between portfolio sizes and the learning ability of DDQN agents: the larger k is, the smaller the hitting ratios. In the second set of experiments, we investigate further the impact of portfolio sizes and problem sizes on DDQN's learning behaviors. We train DDQN agents on `evenly_spread` with a wider set of portfolio sizes $k \in \{3, 4, 5, 6, 7, 8, 10, 15, 20\}$ and with two problem sizes $n \in \{50, 100\}$. For $n = 100$, we expect it to be more difficult for the RL agent to learn due to the larger episode lengths, thus, the training budget is increased to 1.4 million time steps. As shown in Figure 7, DDQN hitting ratios decrease drastically as k increases. For $n = 100$ and $k \geq 7$, the hitting ratios are very close to zero. In fact, the performance of the learned policies for $n = 100$ and $k \in \{15, 20\}$ is no longer competitive to the optimal ones, as shown in Figure 8. Looking into the detailed progress of each RL run, we find that for $k = 7$, the agent barely hits the optimal policies (only 2 times over 750 evaluations), and for $k = 15$, it has zero hitting rate.

The results so far indicate that we reach the learning limit of DDQN with the given setting. To confirm this hypothesis, we repeat

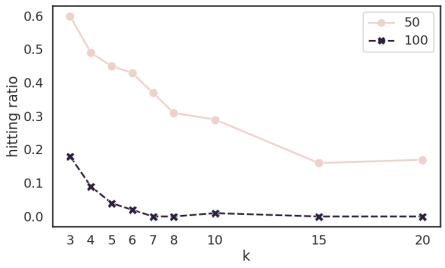


Figure 7: Hitting ratios of DDQN on `evenly_spread`, with $n \in \{50, 100\}$ and $k \in \{3, 4, 5, 6, 7, 8, 10, 15, 20\}$.

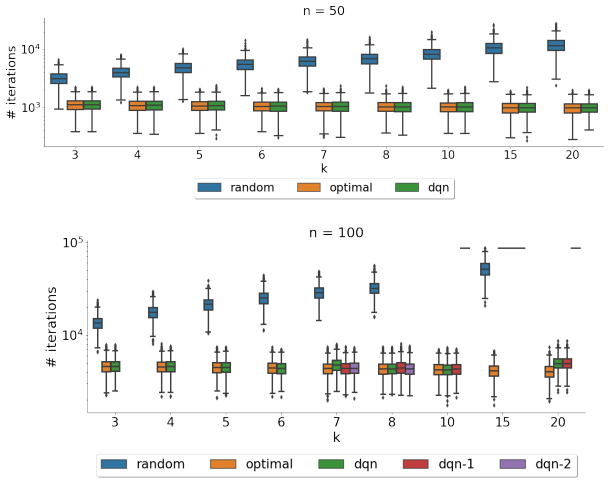


Figure 8: Performance of DDQN on `evenly_spread` setting, with $k \in \{3, 4, 5, 6, 7, 8, 10, 15, 20\}$ and $n \in \{50, 100\}$. DDQN runs failing to learn are marked with a straight line.

the DDQN training two more times for each $k \geq 7$ and $n = 100$. As shown in Figure 8, for $n = 100$ and all $k \geq 10$, there is at least one of three DDQN training runs where the agent does not learn anything, i.e., there is no progress in the entire training process.

Last, we investigate further the impact of problem dimension on the learning limit of DDQN. We train 3 DDQN agents for each pair of $n \in \{150, 200\}$ and $k \in \{3, 4, 5\}$, with a budget of 1.4 million steps and a walltime limit of 48 hours. Within the time limit, each DDQN agent can only reach 400 000 and 250 000 time steps for $n = 150$ and $n = 200$, respectively, since the length of each evaluation episode increases quadratically with n. Figure 9 shows the number of times each agent reaches the performance of the optimal policies during the entire training. These results indicate that $n = 200$ and $k = 5$ is the final limit of our DDQN agent with the chosen hyperparameters, as neither of the three runs can get close to the optimal policy.

5 CONCLUSION AND OUTLOOK

We suggested the optimization of the LeadingOnes problem via the $(1 + 1)$ RLS with fitness-dependent control policies as a benchmark

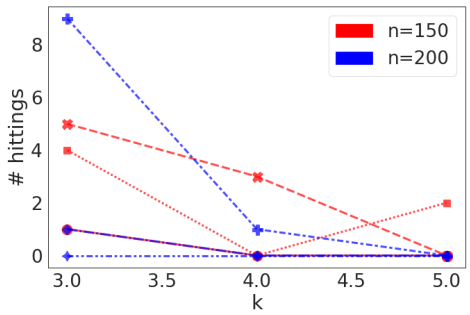


Figure 9: #times DDQN reaches performance of the optimal policy on `evenly_spread`, with $n \in \{150, 200\}$ and $k \in \{3, 4, 5\}$. Linestyles indicate individual runs with different seeds.

problem in the context of dynamic algorithm configuration (DAC). This problem setting is theoretically very well understood, to the point that we could easily extend in this work the base case with full parameter portfolio $[1..n]$ to settings in which the search radii have to be chosen from a restricted portfolio $\mathcal{K} \subsetneq [1..n]$. That is, we can compute optimal control policies for any given combination of problem dimension n and parameter portfolio $\mathcal{K}$. This allows us to create numerous problem instances of different size, which can be leveraged to gain structural insight into the behavior of DAC techniques. Empirically, we showed that DDQN efficiently learns optimal policies for the smaller LeadingOnes instances. We also explored the settings at which DDQN with the chosen parameters and budget reaches its limits, in the sense that the learned policy is not close to optimal or even fails to learn entirely.

One way to overcome the limits of DDQN for larger problem and portfolio sizes could be to use AutoRL [50] to optimize its hyperparameters, such as the batch size, discounting factor, exploration strategy, choice of algorithm, or network architecture. Although it is known that RL agents are very brittle with respect to their hyperparameters, their influence on the learning algorithm is not well understood. Our benchmark enables studying the effect of hyperparameters in a principled manner, which potentially allows us to make RL agents more robust and easier to use for dynamic algorithm configuration. A favorable aspect is that the evaluation times of the LeadingOnes benchmarks are very small, making a systematic investigation on the learning ability of RL agents computationally affordable. In fact, we can reduce the evaluation times further if we replace the actual training process by a simulation that draws the rewards from the well understood reward distribution.

Since we understand the distribution of the reward function perfectly well, no matter the problem dimension, the state, nor the played action (essentially captured by equation (1)), we believe that it is feasible to extend recent theoretical investigations of static algorithm configuration [30] to the more general DAC setting.

Regarding the DAC setting, we did not exploit the full power of DAC, as we trained and tested on the same problem instances and did not aim to derive policies that can be transferred to instances that are not part of the training set, as is classically done in algorithm configuration. Given the promising results of the DDQN agents, a reasonable next step is to investigate the generalization ability of this approach with respect to problem dimension or with respect to the portfolio. Once established, the next step are then to aim for generalizability across different problems, e.g., via a configurable benchmark generator that provides a good fit between problem representation and characteristics. The W-model [65] could be a reasonable playground for first steps in this direction. We note that generalization is an understudied topic in deep RL [41], where DAC and our proposed benchmark can help to advance the field.

Another idea we are keen on exploring is to incorporate other state information into the policy of the $(1+1)$ RLS than just the fitness. For example, for LeadingOnes, Buzdalov and Buzdalova [12] show that adding information about the number of correct bits in the tail allows more efficient control policies. When considering a good configuration of DDQN, this approach could also be applied in order to derive approximately optimal policies for scenarios of state information for which no theoretical guarantees are known.

We emphasize that we investigated the new benchmarks for DAC only, but they are equally interesting for the parameter control setting. Techniques that model parameter control as a multi-armed bandit problem [e.g. 15, 21, 29] can be straightforwardly applied to our benchmarks, as they typically require finite parameter portfolios. We also do not see greater obstacles to adjust other strategies, such as self-adaptive or self-adjusting parameter control mechanisms [26], although the parameter encoding and update strategies may need to be redesigned to account for the restricted portfolio.

We hope that our work initiates a fruitful exchange of benchmarks between parameter control and dynamic algorithm configuration. With the growing literature on parameter control [38] and its theoretical analysis [19], we wish to provide other use-cases with a known ground truth. However, settings for which we have such detailed knowledge as for LeadingOnes are very rare. Even for OneMax, the "drosophila of evolutionary computation" [28], the optimal mutation rates of the $(1+1)$ RLS and the $(1+1)$ evolutionary algorithm are known only in approximate terms [22] or for specific problem dimensions [11, 13, 14]. We believe that an active exchange of theoretically and automatically found policies will benefit both sides: empirical results provide guidance or inspiration for theoretical analyses, and theoretical results can be used as benchmarks with ground truth, as demonstrated in this work.

ACKNOWLEDGMENTS

André Biedenkapp and Frank Hutter acknowledge funding by the Robert Bosch GmbH. Nguyen Dang is a Leverhulme Early Career Fellow. This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 945298-ParisRegion-FP. It is also supported by the Paris Île-de-France region, via the DIM RFSI AlgoSelect project and is partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No. 952215. The authors acknowledge the HPCaVe computing platform of Sorbonne Université for providing computational resources to this research project. The collaboration leading to this work was initiated at the 2020 Lorentz Center workshop "Benchmarked: Optimization Meets Machine Learning".

REFERENCES

[1] Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. 2019. The query complexity of a permutation-based variant of Mastermind. *Discrete Applied Mathematics* 260 (2019), 28–50. https://doi.org/10.1016/j.dam.2019.01.007

[2] Aldeida Aleti and Irene Moser. 2016. A Systematic Literature Review of Adaptive Parameter Control Methods for Evolutionary Algorithms. *Comput. Surveys* 49 (2016), 56:1–56:35.

[3] Thomas Bäck. 1998. An Overview of Parameter Control Methods by Self-Adaption in Evolutionary Algorithms. *Fundam. Informaticae* 35, 1-4 (1998), 51–66. https://doi.org/10.3233/FI-1998-35123404

[4] Roberto Battiti, Mauro Brunato, and Franco Mascia. 2008. *Reactive search and intelligent optimization.* Vol. 45. Springer Science & Business Media.

[5] Roberto Battiti and Paolo Campigotto. 2012. An Investigation of Reinforcement Learning for Reactive Search Optimization. In *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion (Eds.). Springer, 131–160.

[6] André Biedenkapp, H. Furkan Bozkurt, Theresa Eimer, Frank Hutter, and Marius Lindauer. 2020. Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In *Proc. of European Conference on Artificial Intelligence (ECAI'20) (Frontiers in Artificial Intelligence and Applications, Vol. 325).* IOS Press, 427–434. https://doi.org/10.3233/FAIA200122

[7] André Biedenkapp, Nguyen Dang, Martin S. Krejca, Frank Hutter, and Carola Doerr. 2022. Code and data repository of this paper. https://github.com/ndangtt/LeadingOnesDAC.

[8] Mauro Birattari. 2009. *Tuning Metaheuristics - A Machine Learning Perspective.* Studies in Computational Intelligence, Vol. 197. Springer. https://doi.org/10.1007/978-3-642-00483-4

[9] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal Fixed and Adaptive Mutation Rates for the LeadingOnes Problem. In *Proc. of Parallel Problem Solving from Nature (PPSN'10) (LNCS, Vol. 6238).* Springer, 1–10.

[10] Edmund K. Burke, Michel Gendreau, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: a survey of the state of the art. *J. Oper. Res. Soc.* 64, 12 (2013), 1695–1724. https://doi.org/10.1057/jors.2013.71

[11] Nathan Buskulic and Carola Doerr. 2021. Maximizing Drift Is Not Optimal for Solving OneMax. *Evol. Comput.* 29, 4 (2021), 521–541. https://doi.org/10.1162/evco_a_00290

[12] Maxim Buzdalov and Arina Buzdalova. 2015. Can OneMax help optimizing LeadingOnes using the EA+RL method?. In *Proc. of Congress on Evolutionary Computation (CEC'15).* IEEE, 1762–1768. https://doi.org/10.1109/CEC.2015.7257100

[13] Maxim Buzdalov and Carola Doerr. 2020. Optimal Mutation Rates for the $(1+\lambda)$ EA on OneMax. In *Proc. of Parallel Problem Solving from Nature (PPSN'20) (LNCS, Vol. 12270).* Springer, 574–587. https://doi.org/10.1007/978-3-030-58115-2_40

[14] Maxim Buzdalov and Carola Doerr. 2021. Optimal static mutation strength distributions for the $(1+\lambda)$ evolutionary algorithm on OneMax. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'21).* ACM, 660–668. https://doi.org/10.1145/3449639.3459389

[15] Luís Da Costa, Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. 2008. Adaptive operator selection with dynamic multi-armed bandits. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'08).* ACM, 913–920.

[16] Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. 2016. Learning Step Size Controllers for Robust Neural Network Training, See [55].

[17] Benjamin Doerr. 2019. Analyzing randomized search heuristics via stochastic domination. *Theoretical Computer Science* 773 (2019), 115–137. https://doi.org/10.1016/j.tcs.2018.09.024

[18] Benjamin Doerr and Carola Doerr. 2018. Optimal Static and Self-Adjusting Parameter Choices for the (1+(λ,λ)) Genetic Algorithm. *Algorithmica* 80 (2018), 1658–1709. https://doi.org/10.1007/s00453-017-0354-9

[19] Benjamin Doerr and Carola Doerr. 2020. Theory of Parameter Control Mechanisms for Discrete Black-Box Optimization: Provable Performance Gains Through Dynamic Parameter Choices. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization.* Springer, 271–321.

[20] Benjamin Doerr, Carola Doerr, and Johannes Lengler. 2021. Self-Adjusting Mutation Rates with Provably Optimal Success Rules. *Algorithmica* 83, 10 (2021), 3108–3147. https://doi.org/10.1007/s00453-021-00854-3 Available at https://arxiv.org/abs/1902.02588.

[21] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. k-Bit Mutation with Self-Adjusting k Outperforms Standard Bit Mutation. In *Proc. of Parallel Problem Solving from Nature (PPSN'16) (LNCS, Vol. 9921).* Springer, 824–834. https://doi.org/10.1007/978-3-319-45823-6_77

[22] Benjamin Doerr, Carola Doerr, and Jing Yang. 2020. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science* 801 (2020), 1–34. https://doi.org/10.1016/j.tcs.2019.06.014

[23] Benjamin Doerr, Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2018. On the runtime analysis of selection hyper-heuristics with adaptive learning periods. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18).* ACM, 1015–1022. https://doi.org/10.1145/3205455.3205611

[24] Carola Doerr and Johannes Lengler. 2018. The (1+1) Elitist Black-Box Complexity of LeadingOnes. *Algorithmica* 80, 5 (2018), 1579–1603. https://doi.org/10.1007/s00453-017-0304-6 Also available at https://arxiv.org/abs/1604.02355.

[25] Carola Doerr and Markus Wagner. 2018. Simple on-the-fly parameter selection mechanisms for two classical discrete black-box optimization benchmark problems. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18).* ACM, 943–950. https://doi.org/10.1145/3205455.3205560

[26] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 3 (1999), 124–141.

[27] Theresa Eimer, André Biedenkapp, Maximilian Reimer, Steven Adriaensen, Frank Hutter, and Marius Lindauer. 2021. DACBench: A Benchmark Library for Dynamic Algorithm Configuration. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI'21).* ijcai.org, 1668–1674. https://doi.org/10.24963/ijcai.2021/230

[28] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. 2008. Extreme Value Based Adaptive Operator Selection. In *Proc. of Parallel Problem Solving from Nature (PPSN'08) (LNCS, Vol. 5199).* Springer, 175–184.

[29] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. 2010. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence* 60 (2010), 25–64. https://doi.org/10.1007/s10472-010-9213-y

[30] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. 2022. On the impact of the performance metric on efficient algorithm configuration. *Artif. Intell.* 303 (2022), 103629. https://doi.org/10.1016/j.artint.2021.103629

[31] Assaf Hallak, Dotan Di Castro, and Shie Mannor. 2015. Contextual Markov Decision Processes. *CoRR* abs/1502.02259 (2015). http://arxiv.org/abs/1502.02259

[32] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (2001), 159–195. https://doi.org/10.1162/106365601750190398

[33] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second Conference on Artificial Intelligence (AAAI'18)*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 3207–3214.

[34] Holger H. Hoos. 2012. Automated Algorithm Configuration and Parameter Tuning. In *Autonomous Search*, Youssef Hamadi, Éric Monfroy, and Frédéric Saubion (Eds.). Springer, 37–71. https://doi.org/10.1007/978-3-642-21434-9_3

[35] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research* 36 (2009), 267–306.

[36] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2019. *Automated Machine Learning - Methods, Systems, Challenges.* Springer. https://doi.org/10.1007/978-3-030-05318-5

[37] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Population Based Training of Neural Networks. *arXiv:1711.09846 [cs.LG]* (2017).

[38] Giorgos Karafotias, Mark Hoogendoorn, and A.E. Eiben. 2015. Parameter Control in Evolutionary Algorithms: Trends and Challenges. *IEEE Transactions on Evolutionary Computation* 19 (2015), 167–187.

[39] Giorgos Karafotias, Selmar K. Smit, and A. E. Eiben. 2012. A Generic Approach to Parameter Control. In *Proc. of Applications of Evolutionary Computation (EvoApplications'12) (LNCS, Vol. 7248).* Springer, 366–375. https://doi.org/10.1007/978-3-642-29178-4_37

[40] Eric Kee, Sarah Airey, and Walling Cyre. 2001. An Adaptive Genetic Algorithm. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'01).* Morgan Kaufmann, 391–397. https://doi.org/10.5555/2955239.2955303

[41] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. 2021. A Survey of Generalisation in Deep Reinforcement Learning. *arXiv:2111.09794 [cs.LG]* (2021).

[42] Scott Kirkpatrick, C. D. Gelatt, and Mario P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220 (1983), 671–680.

[43] Michail G. Lagoudakis and Michael L. Littman. 2000. Algorithm Selection using Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)*, Pat Langley (Ed.). Morgan Kaufmann Publishers, 511–518.

[44] Michail G. Lagoudakis and Michael L. Littman. 2001. Learning to Select Branching Rules in the DPLL Procedure for Satisfiability. *Electronic Notes in Discrete Mathematics* 9 (2001), 344–359.

[45] Per Kristian Lehre and Carsten Witt. 2012. Black-Box Search by Unbiased Variation. *Algorithmica* 64 (2012), 623–642.

[46] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2020. Simple Hyper-Heuristics Control the Neighbourhood Size of Randomised Local Search Optimally for LeadingOnes. *Evol. Comput.* 28, 3 (2020), 437–461. https://doi.org/10.1162/evco_a_00258

[47] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *Proceedings of the International Conference on Learning Representations (ICLR'17).* Published online: iclr.cc.

[48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[49] Jack Parker-Holder, Vu Nguyen, and Stephen J. Roberts. 2020. Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits. In *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). Curran Associates.

[50] Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. 2022. Automated Reinforcement Learning (AutoRL): A Survey and Open Problems. *CoRR* abs/2201.03916 (2022). arXiv:2201.03916 https://arxiv.org/abs/2201.03916

[51] James E. Pettinger and Richard M. Everson. 2002. Controlling Genetic Algorithms with Reinforcement Learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, W. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. Potter, A. Schultz, J. Miller, E. Burke, and N. Jonoska (Eds.). Morgan Kaufmann Publishers, 692.

[52] Ingo Rechenberg. 1973. *Evolutionsstrategie.* Friedrich Fromman Verlag (Günther Holzboog KG), Stuttgart.

[53] John R. Rice. 1976. The Algorithm Selection Problem. *Advances in Computers* 15 (1976), 65–118.

[54] Yoshitaka Sakurai, Kouhei Takada, Takashi Kawabe, and Setsuo Tsuruta. 2010. A Method to Control Parameters of Evolutionary Algorithms by Using Reinforcement Learning. In *Proceedings of Sixth International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*, K. Yétongnon, A. Dipanda, and R. Chbeir (Eds.). IEEE Computer Society, 74–79.

[55] D. Schuurmans and M. Wellman (Eds.). 2016. *Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16).* AAAI Press.

[56] Gresa Shala, André Biedenkapp, Noor Awad, Steven Adriaensen, Marius Lindauer, and Frank Hutter. 2020. Learning Step-Size Adaptation in CMA-ES. In *Proceedings of the Sixteenth International Conference on Parallel Problem Solving from Nature (PPSN'20) (Lecture Notes in Computer Science).* Springer, 691–706.

[57] Mudita Sharma, Alexandros Komninos, Manuel López-Ibáñez, and Dimitar Kazakov. 2019. Deep Reinforcement Learning-Based Parameter Control in Differential Evolution. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19).* ACM, 709–717. https://doi.org/10.1145/3321707.3321813

[58] Selmar K. Smit and A. E. Eiben. 2009. Comparing parameter tuning methods for evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'09).* IEEE, 399–406. https://doi.org/10.1109/CEC.2009.4982974

[59] David Speck, André Biedenkapp, Frank Hutter, Robert Mattmüller, and Marius Lindauer. 2021. Learning Heuristic Selection with Dynamic Algorithm Configuration. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21)*, H. H. Zhuo, Q. Yang, M. Do, R. Goldman, S. Biundo, and M. Katz (Eds.). AAAI, 597–605.

[60] Dirk Sudholt. 2013. A New Method for Lower Bounds on the Running Time of Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation* 17 (2013), 418–435.

[61] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement learning - an introduction.* MIT Press. https://www.worldcat.org/oclc/37293240

[62] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning, See [55], 2094–2100.

[63] Diederick Vermetten, Sander van Rijn, Thomas Bäck, and Carola Doerr. 2019. Online selection of CMA-ES variants. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19).* ACM, 951–959. https://doi.org/10.1145/3321707.3321803

[64] Christopher. J. C. H. Watkins. 1989. *Learning from Delayed Rewards.* Ph. D. Dissertation. King's College, Cambridge, United Kingdom.

[65] Thomas Weise, Yan Chen, Xinlu Li, and Zhize Wu. 2020. Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms. *Applied Soft Computing* 92 (2020), 106269. https://doi.org/10.1016/j.asoc.2020.106269

Part V

Improving RL From the Lens of DAC

CHAPTER 11

TempoRL: Learning When to Act

The content of this chapter has been published as:

> A. Biedenkapp, R. Rajan, F. Hutter, and M. Lindauer (2021). "TempoRL: Learning When to Act". In: *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 914–924.

Project Idea. The idea was proposed by André Biedenkapp based on the observation that hyperparameter values do not necessarily need to be adapted at every time step. Marius Lindauer proposed to use TempoRL not only on discrete action spaces but also on continuous action spaces. André Biedenkapp proposed the different neural architectures used in the deep RL settings with feedback by Marius Lindauer.

Implementation and experimentation. Implementation was led by André Biedenkapp. Raghu Rajan aided in the implementation, debugging, and configuration of RL agents on Atari environments. André Biedenkapp conducted the experiments with support by Raghu Rajan for the Atari environments.

Paper writing. The first draft of the paper was written by André Biedenkapp. Marius Lindauer provided feedback on better structuring of the paper and aided in formally defining skip MDPs. Frank Hutter and Raghu Rajan provided feedback on this version of the paper. The paper was finalized by André Biedenkapp.

TempoRL: Learning When to Act

André Biedenkapp [1] **Raghu Rajan** [1] **Frank Hutter** [1 2] **Marius Lindauer** [3]

Abstract

Reinforcement learning is a powerful approach to learn behaviour through interactions with an environment. However, behaviours are usually learned in a purely reactive fashion, where an appropriate action is selected based on an observation. In this form, it is challenging to learn *when* it is necessary to execute new decisions. This makes learning inefficient, especially in environments that need various degrees of fine and coarse control. To address this, we propose a proactive setting in which the agent not only selects an action in a state but also for how long to commit to that action. Our TempoRL approach introduces skip connections between states and learns a skip-policy for repeating the same action along these skips. We demonstrate the effectiveness of TempoRL on a variety of traditional and deep RL environments, showing that our approach is capable of learning successful policies up to an order of magnitude faster than vanilla $\mathcal{Q}$-learning.

1. Introduction

Although reinforcement learning (RL) has celebrated many successes in the recent years (see e.g., Mnih et al., 2015; Lillicrap et al., 2016; Baker et al., 2020), in its classical form it is limited to learning policies in a mostly reactive fashion, i.e., observe a state and react to that state with an action. Guided by the reward signal, policies that are learned in such a way can decide which action is expected to yield maximal long-term rewards. However, these policies generally do not learn *when a new decision has to be made*. A more proactive way of learning, in which agents proactively commit to playing an action for multiple steps could further improve RL by (i) potentially providing better exploration compared to common one-step exploration; (ii) faster learning as proactive policies provide a form of temporal abstraction by requiring fewer decisions; (iii) explainability as learned agents can indicate when they expect new decisions are required.

Temporal abstractions are a common way to simplify learning of policies with potentially long action sequences. Typically, the temporal abstraction is learned on the highest level of a hierarchy and the required behaviour on a lower level (see e.g. Sutton et al., 1999; Eysenbach et al., 2019). For example, on the highest level a *goal policy* learns which states are necessarily visited and on the lower level the *behaviour* to reach goals is learned. Spacing goals far apart still requires to learn complex behaviour policies whereas a narrow goal spacing requires to learn complex goal policies. Another form of temporal abstraction is to use actions that work at different time-scales (Precup et al., 1998). Take for example an agent that is tasked with moving an object. On the highest level the agent would follow a policy with abstract actions, such as *pick-up object*, *move object*, *put-down object*, whereas on the lower level actions could directly control actuators to perform the abstract actions.

Such hierarchical approaches are still reactive, but instead of reacting to an observation on only one level, reactions are learned on multiple levels. Though these approaches might allow us to learn *which* states are necessarily traversed in the environment, they do not enable us to learn *when* a new decision has to be made on the behaviour level.

In this work, we propose an alternative approach: a proactive view on learning policies that allows us to jointly learn a behaviour and how long to carry out that behaviour. To this end, we re-examine the relationship between agent and environment, and the dependency on time. This allows us to introduce *skip connections* for an environment. These skip connections do not change the optimal policy or state-action-values but allow us to propagate information much faster. We demonstrate the effectiveness of our method, which we dub TEMPORL with tabular and deep function approximation on a variety of environments with discrete and continuous action spaces. Our contributions are:

1. We propose a proactive alternative to classical RL.
2. We introduce skip-connections for MDPs by playing an action for several consecutive states, which leads to faster propagation of information about future rewards.

[1]Department of Computer Science, University of Freiburg, Germany [2]BCAI, Renningen, Germany [3]Information Processing Institute (tnt), Leibniz University Hannover, Germany. Correspondence to: André Biedenkapp <biedenka@cs.uni-freiburg.de>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

3. We propose a mechanism based on a hierarchy for learning when to make new decisions through the use of skip-connections.
4. On classical and deep RL benchmarks, we show that TEMPORL outperforms plain DQN, DAR and FiGAR both in terms of learning speed and sometimes even by converging to better policies.

2. Related Work

A common framework for temporal abstraction in RL is the options framework (Precup et al., 1998; Sutton et al., 1999; Stolle & Precup, 2002; Bacon et al., 2017; Harutyunyan et al., 2018; Mankowitz et al., 2018; Khetarpal & Precup, 2019). Options are triples $\langle \mathcal{I}, \pi, \beta \rangle$ where $\mathcal{I}$ is the set of admissible states that defines in which states the option can be played; π is the policy the option follows when it is played; and β is a random variable that determines when an option is terminated. In contrast to our work, options require a lot of prior knowledge about the environment to determine the set of admissible states, as well as the option policies themselves. However, Chaganty et al. (2012) proposed to learn options based on observed connectedness of states. Similarly, SoRB (Eysenbach et al., 2019) uses data from the replay buffer to build a connectedness graph, which allows to query sub-goals on long trajectories. Further work on discovering options paid attention to the termination criterion, learning persistent options (Harb et al., 2018) and meaningful termination criteria (Vezhnevets et al., 2016; Harutyunyan et al., 2019).

Similarly, in AI planning macro actions provide temporal abstractions. However, macro actions are not always applicable as some actions can be locked. Chrpa & Vallati (2019) propose to learn when macro actions become available again, allowing them to identify non-trivial activities. For various problem domains of AI planning, varieties of useful macro actions are known and selecting which macro actions to consider is not trivial. Vallati et al. (2019) propose a macro action selection mechanism that selects which macro actions should be considered for new problems. Further, Nasiriany et al. (2019) show that goal-conditioned policies learned with RL can be incorporated into planning. With complex state observations goal states are difficult to define.

An important element in DQN's success in tackling various Atari games (Mnih et al., 2015) is due to the use of *frame skipping* (Bellemare et al., 2013). Thereby the agent skips over a few states, always playing the same action, before making a new decision. Without the use of frame skipping, the change between successive observations is small and would have required more observations to learn the same policy. Tuning the *skip-size* can additionally improve performance (Braylan et al., 2015; Khan et al., 2019). A similar line of research focuses on learning persistent policies which always act after a static, fixed time-step for one-dimensional (Metelli et al., 2020) and multi-dimensional actions (Lee et al., 2020). However, static skip-sizes might not be ideal. Dabney et al. (2020) demonstrated that temporally extended ϵ-greedy schedules improve exploration and thereby performance in sparse-reward environments while performing close to vanilla ϵ-greedy exploration on dense-reward environments.

Different techniques have been proposed to handle continuous time environments (Doya, 2000; Tiganj et al., 2017). Recently, Huang et al. (2019) proposed to use *Markov Jump Processes* (MJPs). MJPs are designed to study optimal control in MDPs where observations have an associated cost. The goal then is to balance the costs of observations and actions to act in an optimal manner with respect to total cost. Their analysis demonstrated that frequent observations are necessary in regions where an optimal action might change rapidly, while in areas of infrequent change, fewer observations are sufficient. In contrast to ours, this formalism strictly prohibits observations of the skipped transitions to save observation costs and thus losing a lot of information, which otherwise could be used to learn how to act while simultaneously learning when new decisions are required.

Schoknecht & Riedmiller (2002; 2003) demonstrated that learning with multi-step actions can significantly speed up RL. Relatedly, Lakshminarayanan et al. (2017) proposed *DAR*, a $\mathcal{Q}$-network with multiple output heads per action to handle different repetition lengths, drastically increasing the action space but improving learning. In contrast to that, Sharma et al. (2017) proposed *FiGAR*, a framework that jointly learns an action policy and a second repetition policy that decides how often to repeat an action. Crucially, its repetition policy is not conditioned on the chosen action resulting in independent repetition and behaviour actions. The polices are learned together through a joint loss. Thus, counter to our work, the repetition policy only learns which repetition length works well on average for all actions. Further, FiGAR requires modification to the training method of a base agent to accommodate the repetition policy. When evaluating our method in the context of DQN, we compare against DAR and in the context of DDPG against FiGAR as they were originally developed and evaluated on these agent types. The appendix, code and experiment results are available at `github.com/automl/TempoRL`.

3. TempoRL

We begin this section by introducing skip connections into MDPs, propagating information about expected future rewards faster. We then introduce a novel learning mechanism that makes use of a hierarchy to learn a policy that is capable of not only learning which action to take, but also *when* a new action has to be chosen.

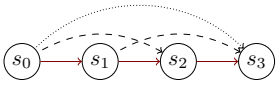


Figure 1. Example transitions with skip of length three (drawn with $\cdots$). At the same time we can also observe shorter skips of length two (- - -) and normal steps, i.e. skips of length one ($-$).

3.1. Temporal Abstraction through Skip MDPs

It is possible to make use of contextual information in MDPs (Hallak et al., 2015; Modi et al., 2018; Biedenkapp et al., 2020). To this end, we contextualize an existing MDP $\mathcal{M}$ to allow for skip connections as $\mathcal{M}_\mathcal{C} \coloneqq \{\mathcal{M}_c\}_{c\in\mathcal{C}}$ with $\mathcal{M}_c \coloneqq \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_c, \mathcal{R}_c \rangle$. Akin to options, a skip-connection c is a triple $\langle s, a, j \rangle$, where s is the starting state for a skip transition (and not a set of states as in the options framework); a is the action that is executed when skipping through the MDP; and j is the skip-length, where $a \in \mathcal{A}$, $s \in \mathcal{S}$ and $j \in \mathcal{J} = \{1, \dots, J\}$. This context to the MDP induces different MDPs with shared state and action spaces ($\mathcal{S}$, $\mathcal{A}$), but different transitions $\mathcal{P}_c$ and reward functions $\mathcal{R}_c$ to account for the introduced skips.

In practice however, the transition and reward functions are unknown and do not allow to easily insert skips. Nevertheless, as we make use of action repetition, we can simulate a skip connection. A skip connects two states s and s' iff state s' is reachable from state s by repeating action a j-times. This gives us the following skip transition function:

$$\mathcal{P}_c(s, a, s') = \begin{cases} \prod_{k=0}^{j-1} \mathcal{P}^a_{s_k s_{k+1}} & \text{if reachable} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

with s_k and s_{k+1} the states traversed by playing action a for the kth time, and with $s_0 = s$ and $s_j = s'$. This change in the transition function is reflected accordingly in the reward:

$$\mathcal{R}_c(s, a, s') = \begin{cases} \sum_{k=0}^{j-1} \gamma^k \mathcal{R}^a_{s_k s_{k+1}} & \text{if reachable} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Thus, for skips of length 1 we recover the original transition function $\mathcal{P}_{\langle s,a,1\rangle}(s, a, s') = \mathcal{P}^a_{ss'}$ as well as the original reward function $\mathcal{R}_{\langle s,a,1\rangle}(s, a, s') = \mathcal{R}^a_{ss'}$. The goal with skip-MDPs is to find an optimal skip policy $\pi_J \colon \mathcal{S} \times \mathcal{A} \mapsto \mathcal{J}$, i.e., a policy that takes a state and a behaviour action as input and maps to a skip value that maximally reduces the total required number of decisions to reach the optimal reward. Thus, similar to skip-connections in neural networks, skip MDPs allow us to propagate information about future rewards much more quickly and enables us to determine *when* it becomes beneficial to switch actions.

3.2. Learning When to Make Decisions

In order to learn using skip connections we need a new mechanism that selects which skip connection to use. In order to facilitate this, we propose using a hierarchy in which a *behaviour* policy determines the action a to be played given the current state s, and a *skip* policy determines how long to commit to this behaviour.

To learn the behaviour, we can make use of classical $\mathcal{Q}$-learning, where the $\mathcal{Q}$-function gives a mapping of expected future rewards when playing action a in state s_t at time t and continuing to follow the behaviour policy π thereafter.

$$\mathcal{Q}^\pi(s, a) \coloneqq \mathbb{E}\left[r_t + \gamma \mathcal{Q}^\pi(s_{t+1}, a_{t+1}) | s = s_t, a\right] \tag{3}$$

To learn to skip, we first have to define a *skip-action space* that determines all possible lengths of skip-connections, e.g., $j \in \{1, 2, \dots, J\}$. To learn the value of a skip we can make use of n-step $\mathcal{Q}$-learning with the condition that, at each step of the j steps, the action stays the same.

$$\mathcal{Q}^{\pi_J}(s, j|a) \coloneqq \mathbb{E}\left[\sum_{k=0}^{j-1} \gamma^k r_{t+k} + \gamma^j \mathcal{Q}^\pi(s_{t+j}, a_{t+j}) | s = s_t, a, j\right] \tag{4}$$

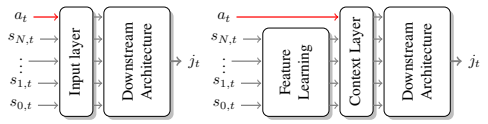
We call this a flat hierarchy since the behaviour and the skip policy have to always make decisions at the same time-step; however, the behaviour policy has to be queried before the skip policy. Once we have determined both the action a and the skip-length j we want to perform, we execute this action for j steps. We can then use standard temporal difference updates to update the behaviour and skip $\mathcal{Q}$-functions with all one-step observations and the overarching skip-observation. Note that the skip $\mathcal{Q}$-function can also be conditioned on continuous actions if the behaviour policy can handle continuous action-spaces.

One interesting observation regarding this learning scheme is that, when playing skip action j, we are able to also observe all smaller skip transitions for all intermediate steps. Figure 1 gives a visual representation. Specifically, we can directly see that, when executing a skip of length j, we can observe and learn from $\frac{j\cdot(j+1)}{2}$ skip-transitions in total. As we observe all intermediate steps, we can use this trajectory of transitions to build a local connectedness graph (similar to Figure 1) from which we can look up all skip-connections. This allows us to efficiently learn the values of multiple skips, i.e. the action value at different time-resolutions. For pseudo-code and more details we refer to Appendix B.

3.3. Learning When to Make Decisions in Deep RL

When using deep function approximation for TEMPORL we have to carefully consider how we parameterize the skip policy. Commonly, in deep RL we do not only deal with featurized states but also with image-based ones. Depending on the state modality we can consider different architectures:

Concatenation The simplest parametrization of our skip-policy assumes that the state of the environment we are



(a) Concat (b) Context

Figure 2. Schematic representations of considered architectures for learning when to make decisions, where a_t is the action coming from a separate behaviour policy.

learning from is featurized, i.e., a state is a vector of individual informative features. In this setting, the skip-policy network can take any architecture deemed appropriate for the environment, where the input is a concatenation of the original state s_t and the chosen behaviour action a_t, i.e., $s'_t = (s_t, a_t)$, see Figure 2a. This allows the skip-policy network to directly learn features that take into account the chosen behaviour action. However, note that this concatenation assumes that the state is already featurized.

Contextualization In deep RL, we often have to learn to act directly from images. In this case, concatenation is not trivially possible. Instead we propose to use the behaviour action as context information further down-stream in the network. Feature learning via convolutions can then progress as normal and the learned high-level features can be concatenated with the action a_t and be used to learn the final skip-value, see Figure 2b.

Shared Weights Concatenation and contextualization learn individual policy networks for the behaviour and skip policies and do not share information between the two. To achieve this we can instead share parts of the networks, e.g., the part of learning higher-level features from images (see Figure 3). This allows us to learn the two policy networks with potentially fewer weights than two completely independently learned networks. In the forward and backward passes, only the shared feature representation with the corresponding output layers are active. Similar to the contextualization, the output layers for the skip-values require the selected action, i.e. the argmax of the action outputs, as additional input.

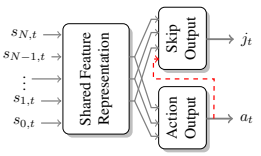


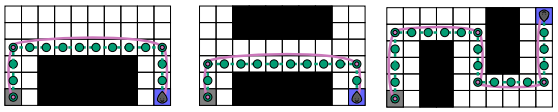
Figure 3. Architecture with shared feature representation for joint learning of when to make a decision and what action to take.

4. Experiments

We evaluated TEMPORL with tabular as well as deep $\mathcal{Q}$-functions. We first give results for the tabular case. All code, the appendix and experiment data including trained policies are available at `github.com/automl/TempoRL`. For details on the used hardware see Appendix C.

4.1. Tabular TempoRL

In this subsection, we describe experiments for a tabular $\mathcal{Q}$-learning implementation that we evaluated on various grid-worlds with sparse rewards (see Figure 4). We first evaluate our approach on the *cliff* environment (see Figure 4a) before evaluating the influence of the exploration schedule on both vanilla and TEMPORL $\mathcal{Q}$-learning, which we refer to as $\mathcal{Q}$ and t-$\mathcal{Q}$, respectively.



(a) Cliff (b) Bridge (c) ZigZag

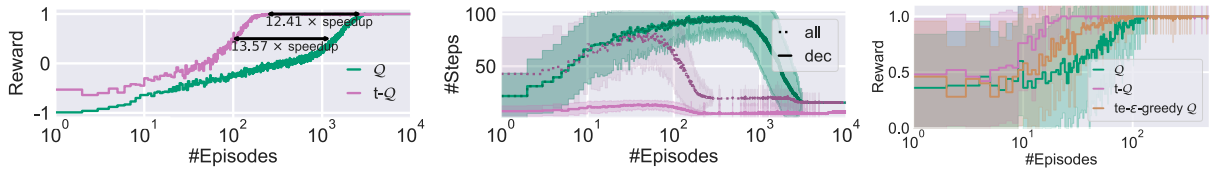
Figure 4. 6×10 Grid Worlds. Agents have to reach a fixed goal state from a fixed start state. Dots represent decision steps of vanilla and TEMPORL $\mathcal{Q}$-learning policies.

Gridworlds All considered environments (see Figure 4) are discrete, deterministic, have sparse rewards and have size 6×10. Falling off a cliff results in a negative reward (-1) and reaching a goal state results in a positive reward $(+1)$. For a more detailed description of the gridworld environments we refer to Appendix D.

For this experiment, we limit our TEMPORL agent to a maximum skip length of $J = 7$; thus, a learned optimal policy requires 4 decision points instead of 3. For evaluations using larger skips we refer to Appendix E. Note that increasing the skip-length improves TEMPORL up to some point, at which it has too many irrelevant skip-actions at its disposal which slightly decreases the performance. We compare the learning speed, in terms of training policies, of our approach to a vanilla $\mathcal{Q}$-learning agent. Both methods are trained for 10 000 episodes using the same ϵ-greedy strategy, where ϵ is linearly decayed from 1.0 to 0.0 over all episodes.

Figure 5a depicts the evaluation performance of both methods. TEMPORL is $13.6\times$ faster than its vanilla counterpart to reach a reward of 0.5, and $12.4\times$ faster to reach a reward of 1.0 (i.e., always reach the goal). Figure 5b shows the number of required steps in the environment, as well as the number of decision steps. TEMPORL is capable of finding a policy that reaches the goal much faster than vanilla $\mathcal{Q}$-learning while requiring far fewer decision steps. Furthermore, TEMPORL recovers the optimal policy quicker than vanilla $\mathcal{Q}$-learning. Lastly we can observe that after having trained for $\approx 6\,000$ episodes, TEMPORL starts to increase the number of decision points. This can be attributed to skip values of an action having converged to the same value and our implementation selecting a random skip as tie-breaker.

Table 1 summarizes the results on all environments in terms of normalized area under the reward curve and number of decisions for three different ϵ-greedy schedules. A reward AUC value closer to 1.0 indicates that the agent was capable



(a) Cliff – Reward per Episode (b) Cliff – Steps per Episode (c) Temporal Exploration

Figure 5. Evaluation performance of tabular $\mathcal{Q}$-learning agents over 100 random seeds. (a) & (b): The agents were trained with a linearly-decaying ϵ-greedy policy on the cliff environment. (a) Achieved reward. (b) Length of executed policy ($\cdots$) and number of decisions (—) made by the policies. (c) Comparison to temporally extended ϵ-greedy exploration (te-ϵ-greedy $\mathcal{Q}$ in plot) on a 23×23 Gridworld (Dabney et al., 2020). t-$\mathcal{Q}$ is our proposed TEMPORL agent. The lines/shaded area represent the mean/standard deviation.

Table 1. Normalized AUC for reward and average number of decision steps. Both agents are trained with the same ϵ schedule.

(a) linearly decaying ϵ-schedule

	Cliff		Bridge		ZigZag	
	$\mathcal{Q}$	t-$\mathcal{Q}$	$\mathcal{Q}$	t-$\mathcal{Q}$	$\mathcal{Q}$	t-$\mathcal{Q}$
$\text{Reward}_{\text{AUC}}$	0.92	**0.99**	0.75	**0.97**	0.57	**0.92**
Decisions	27.9	**5.2**	49.5	**5.0**	83.6	**7.9**

(b) logarithmically decaying ϵ-schedule

$\text{Reward}_{\text{AUC}}$	0.96	**0.99**	0.94	**0.98**	0.90	**0.96**
Decisions	21.7	**4.9**	21.4	**5.3**	35.6	**6.9**

(c) constant $\epsilon = 0.1$

$\text{Reward}_{\text{AUC}}$	**0.99**	**0.99**	0.98	**0.99**	0.95	**0.99**
Decisions	17.1	**5.1**	14.7	**5.2**	27.6	**7.1**

of learning to reach the goal quickly. A lower number of decisions is better as fewer decisions were required to reach the goal, making a policy easier to learn. In view of both metrics, TEMPORL readily outperforms the vanilla agent, learning much faster and requiring far fewer decisions.

Sensitivity to Exploration As the used exploration mechanism can have a dramatic impact on agent performance we evaluated the agents for three commonly used ϵ-greedy exploration schedules. In the cases of linearly and logarithmically decaying schedules, we decay ϵ over all 10 000 training episodes, starting from 1.0 and decaying it to 0 or 10^{-5}, respectively. In the constant case, we set $\epsilon = 0.1$.

As shown in Table 1, maybe not surprisingly, too much (linear) and too little (log) exploration are both detrimental to the agent's performance. However, TEMPORL performs quite robustly even using suboptimal exploration strategies. TEMPORL outperforms its vanilla counterpart in all cases, showing the effectiveness of our proposed method.

Guiding Exploration To demonstrate TEMPORL not only benefits through better exploration but also learning *when* to act, we use the 23×23 Gridworld and agent hyperparameters as introduced by Dabney et al. (2020).

An agent starts in the top center and has to find a goal further down and to the left only getting a reward for reaching the goal within 1000 steps. Temporally-extended exploration (te-ϵ-greedy $\mathcal{Q}$-learning; Dabney et al., 2020) is able to cover a space much better than 1-step exploration. However, it falls short in guiding the agent back to high reward areas. TEMPORL enables an agent to quickly find a successful policy that reach a goal while exploring around such a policy. Figure 5c shows TEMPORL reliably reaches the goal after ≈ 30 episodes. An agent using temporally-extended epsilon greedy exploration does not reliably reach the goal in this time-frame and on average requires twice as many steps.

4.2. Deep TempoRL

In this section, we describe experiments for agents using deep function approximation implemented with PyTorch (Paszke et al., 2019) in version 1.4.0. We begin with experiments on featurized environments before evaluating on environments with image states. We evaluate TEMPORL for DQN with different architectures for the skip $\mathcal{Q}$-function. We compare against *dynamic action repetition* (DAR; Lakshminarayanan et al., 2017) for the DQN experiments and against *Fine grained action repetition* (FiGAR Sharma et al., 2017) for experiments with DDPG.[1]

4.2.1. ADVERSARIAL ENVIRONMENT – DDPG

Setup We chose to first evaluate on OpenAI gyms (Brockman et al., 2016) Pendulum-v0 as it is an adversarial setting where high action repetition is nearly guaranteed to overshoot the balancing point. Thus, agents using action repetition that make mistakes during training will have to spend additional time learning *when* it is necessary to be reactive; a challenge vanilla agents are not faced with. We trained all DDPG agents (Lillicrap et al., 2016) for a total of 3×10^4 training steps and evaluated the agents every 250 training steps. The first 10^3 steps follow a uniform random policy to generate the initial experience. We used Adam (Kingma & Ba, 2015) with PyTorchs default settings.

[1]Neither DAR, nor FiGAR are publicly available and thus we used our own reimplementation available at `github.com/automl/TempoRL`.

Table 2. Average normalized reward AUC for DDPG agents on Pendulum-v0. t-DDPG and FiGAR are evaluated over different maximal skip-lengths for 15 seeds. Corresponding learning curves are given in Appendix F

	t-DDPG							FiGAR						
DDPG	2	4	6	8	10	14	20	2	4	6	8	10	14	20
0.92	0.89	0.89	*0.90*	0.89	0.89	0.89	0.88	*0.76*	0.57	*0.39*	0.31	0.28	0.25	0.24

Agents All actor and critic networks of all DDPG agents consist of two hidden layers with 400 and 300 hidden units respectively. Following (Sharma et al., 2017), FiGAR introduces a second actor network that shares the input layer with the original actor network. The output layer is a softmax layer with J outputs, representing the probability of repeating the action for $j \in \{1, \dots, J\}$ time-steps. Both actor outputs are jointly input to the critic and gradients are directly propagated from the critic through both actors. TEMPORL DDPG (which we refer to as t-DDPG in the following) uses the concatenation architecture which takes the state with the action output of the DDPG actor as input and makes use of the critic's $\mathcal{Q}$-function when learning the skip $\mathcal{Q}$-function. We evaluate t-DDPG and FiGAR on a grid of maximal skip lengths of $\{2, 4, 6, \dots, 20\}$. See Appendix F for implementation details and all used hyperparameters.

Pendulum Table 2 confirms that agents using action repetition indeed are slower in learning successful policies, as reflected by the normalized reward AUC. As FiGAR does not directly inform the skip policy about the chosen repetition value or vica versa, the agent tends to struggle quite a lot in this environment already with only two possible skip-values and is not capable of handling larger maximal skip values. In contrast to that, t-DDPG only slightly lags behind vanilla DDPG and readily adapts to larger skip lengths, by quickly learning to ignore irrelevant skip-values. Further, due to making use of n-step learning, t-DDPG starts out very conservative as large skip values appear to lead to larger negative rewards in the beginning. With more experience however, t-DDPG learns *when* switching between actions becomes advantageous, thereby approximately halving the required decisions (see Appendix F).

4.2.2. FEATURZIED ENVIRONMENTS – DQN

Setup We trained all agents for a total of 10^6 training steps using a constant ϵ-greedy exploration schedule with ϵ set to 0.1. We evaluated all agents every 200 training steps. We used the Adam with a learning rate of 10^{-3} and default parameters as given in PyTorch v1.4.0. For increased learning stability, we implemented all agents using double deep $\mathcal{Q}$ networks (van Hasselt et al., 2016). All agents used a replay buffer with size 10^6 and a discount factor γ of 0.99. The TEMPORL agents used an additional replay buffer of size 10^6 to store observed skip-transitions. We used the MountainCar-v0 and LunarLander-v2 environments. See Appendix G for a detailed description of the environments.

Agents The basic DQN architecture consists of 3 layers, each with 50 hidden units and *ReLu* activation functions. The DAR baseline used the same architecture as the DQN agent but duplicated the output heads twice, each of which is associated with specific repetition values allowing for fine and coarse control. We evaluated possible coarse control values on the grid $\{2, 4, 6, 8, 10\}$, keeping the fine-control value fixed to 1 to allow for actions at every-time step.

For TEMPORL agents not using weight sharing we used the same DQN architecture for both $\mathcal{Q}$-functions. The concatenation architecture used an additional input unit whereas the context architecture added the behaviour action as context at the third layer after using 10 additional hidden units to process the behaviour action. An agent using a weight-sharing architecture shared the first two layers of the DQN architecture and used the third layer of the DQN architecture to compute the behaviour $\mathcal{Q}$-values. The skip-output used 10 hidden units to process the behaviour action and processed this output together with the hidden state of the 2nd layer in a 3rd layer with 60 hidden units. We refer to a DQN using TEMPORL as t-DQN in the following.

Influence of the Skip-Architecture We begin by evaluating the influence of architecture choice on our t-DQN on both environments, before giving a more in-depth analysis on the learning behaviour in the individual environments. To this end, we report the normalized reward AUC for all three proposed architectures and different maximal skip-lengths, see Table 3. Both the concat and context architectures behave similarly on both environments, which is to be expected as they differ very little in setup. Both architectures have an increase in AUC before reaching the best maximal skip-length for the respective environment. The shared architecture, mostly conceptualized for image-based environments, however shows more drastic reactions to choice of J, leading to the best result in the first and to the worst result in the other environment.

MountainCar Tables 3a & 4a depict the performance of the agents for different maximal skip lengths and Figure 6a shows the learning curves of the best TEMPORL architecture as well as the best found DAR agent. On MountainCar the DQN baseline struggles in learning a successful policy, resulting in a small AUC of 0.50 compared to the best result of t-DQN of 0.64. Furthermore, a well tuned DAR baseline,

Table 3. Average normalized reward AUC for different TEMPORL architectures and maximal skip-lengths over 50 seeds. All agents are trained with the same ϵ schedule. Bold faced values give the overall best AUC and cursive values the best per architecture.

(a) MountainCar-v0

Max Skip	2	4	6	8	10
concat	0.469	0.523	0.602	0.626	*0.630*
context	0.429	0.540	0.601	0.608	*0.620*
shared	0.440	0.464	0.592	0.561	***0.644***

(b) LunarLander-v2

Max Skip	2	4	6	8	10
concat	0.855	***0.878***	0.868	0.862	0.830
context	0.858	*0.876*	0.871	0.859	0.837
shared	*0.851*	0.837	0.803	0.769	0.696

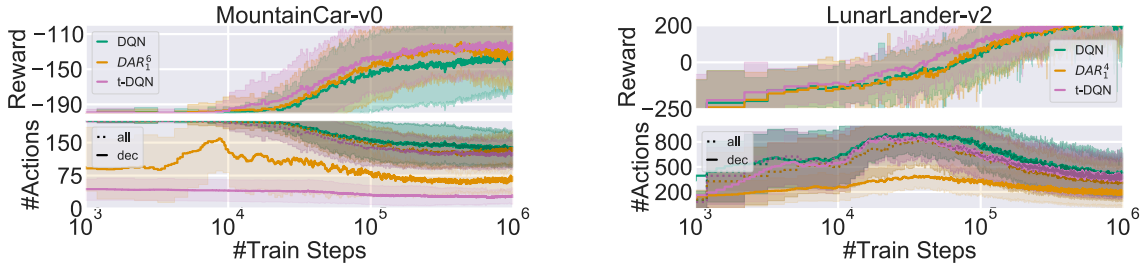


Figure 6. Evaluation performance of deep $\mathcal{Q}$-learning agents on MountainCar-v0 and LunarLander-v2. Solid lines give the mean and the shaded area the standard deviation over 50 random seeds. The sub- and superscripts of DAR give the best found fine and coarse repetition values respectively. t-DQN is our proposed method using the best architecture as reported in Table 3. (top) Achieved rewards. (bottom) Length of executed policy ($\cdots$) and number of decisions (—) made by the policies.

Table 4. Average normalized reward AUC for maximal skip-length of 10 for MountainCar-v0 and 4 for LunarLander-v2 over 50 seeds. All agents are trained with the same ϵ schedule. We show varying $^{max}_{min}$ repetitions for DAR and the best t-DQN architecture (see Table 3). Bold faced values give the overall best AUC and cursive values the best per method which are plotted in Figure 6.

(a) MountainCar-v0

		DAR				
DQN	t-DQN	$^{2}_{1}$	$^{4}_{1}$	$^{6}_{1}$	$^{8}_{1}$	$^{10}_{1}$
0.50	***0.64***	0.43	0.45	*0.60*	0.56	0.56

(b) LunarLander-v2

0.83	***0.88***	0.84	*0.85*	0.81	0.72	0.60

carefully trading off fine control and coarse control results in an AUC of 0.593. Figure 6a shows that DAR learns to trade off both coarse and fine-control. However, as DAR does not know that two output heads correspond to the same action, with different repetition values, DARs reward begins to drop in the end as it learns to overly rely on coarse control. During the whole training procedure the best t-DQN agent and the best DAR agent result in policies that require far fewer decisions, with t-DQN requiring only ≈ 50 decisions per episode reducing the number of decisions by a factor of ≈ 3 compared to vanilla DQN.

LunarLander For such a dense reward there is only a small improvement for t-DQN and a properly tuned DAR agent. Again the t-DQN agent performs best, achieving a slightly higher AUC of 0.88 than the best tuned DAR agent (0.85), see Tables 3b & 4b. Further, Figure 6b shows that, in this setting, t-DQN agents quickly learn to be very reactive, acting nearly at every time-step. Again, DAR can not learn that some output heads apply the same behaviour action for multiple time-steps, preferring coarse over fine control.

4.2.3. ATARI ENVIRONMENTS

Setup We trained all agents for a total of 2.5×10^6 training steps (i.e. only 10 million frames) using a linearly ϵ-greedy exploration schedule over the first 200 000 time-steps with a final fixed ϵ set to 0.01. We evaluated all agents every 10 000 training steps and evaluated for 3 episodes. For increased learning stability we implemented all agents using double deep $\mathcal{Q}$ networks. For DQN we used the architecture of Mnih et al. (2015) which also serves as basis for our shared t-DQN and the DAR architecture. As maximal skip-value we chose 10. A detailed list of hyperparameters is given in Appendix H. Following (Bellemare et al., 2013), we used a frame-skip of 4 to allow for a fair comparison to the base DQN. We used *OpenAI Gym*'s We trained all agents on the games BEAMRIDER, FREEWAY, MSPACMAN, PONG and QBERT.

Learning When to Act in Atari Figure 7 depicts the learning curves as well as the number of steps and decisions. The training behaviour from our TEMPORL agents falls into one of three categories on all evaluated Atari games.

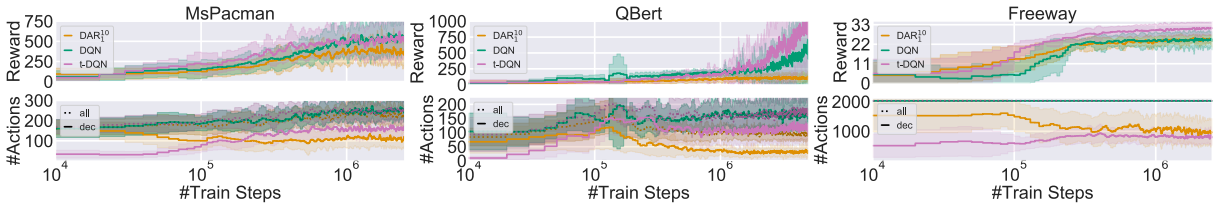


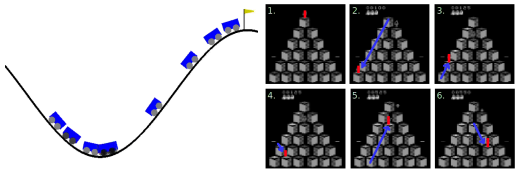
Figure 7. Evaluation performance on Atari environments. Solid lines give the mean and the shaded area the standard deviation over 15 random seeds. (top) Achieved rewards. (bottom) Length of executed policy (· · ·) and number of decisions (—) made by the policies. To make trends more visible, we smooth over a window of width 7.

(i) Our learned t-DQN exhibits a slight improvement in learning speed, on MsPacman and Pong[2] before being caught up by DQN, with both methods converging to the same final reward (see Figures 7a & H1a). Nevertheless, TempoRL learns to make use of different degrees of fine and coarse control to achieve the same performance. For example, a trained proactive TempoRL policy requires roughly 33% fewer decisions. DAR on the other hand, learns to overly rely on the coarse control, leading to far fewer decisions but also worse final performance.

(ii) On QBert the learning performance of our t-DQN lags behind that of vanilla DQN over the first 10^6 steps. Figure 7b (bottom) shows that in the first $\approx 0.5 \times 10^6$ steps, TempoRL first has to learn which skip values are appropriate for Qbert. In the next $\approx 0.5 \times 10^6$ steps, our t-DQN begins to catch up in reward, while using its learned fine and coarse control, before starting to overtake its vanilla counterpart. As it was not immediately clear if this trend would continue after 2.5×10^6 training steps, we continued the experiment for twice as many steps. TempoRL continues to outperform its vanilla counterpart, having learned to trade off different levels of coarse and fine control. The effect of over-reliance of DAR on the coarse control is further amplified on QBert resulting in far worse policies than either vanilla DQN and TempoRL.

(iii) In games such as Freeway and BeamRider (Figures 7c & H1b), we see an immediate benefit to jointly learning *when* and *how* to act through TempoRL. For these games, our t-DQNs begin to learn faster and achieve a better final reward than vanilla DQNs. An extreme example for this is Freeway, where the agents have to control a chicken to cross a busy multi-lane highway as often as possible within a fixed number of frames. To this end one action has to be played predominantly, whereas the other two possible actions are only needed to sometimes avoid an oncoming car. The vanilla DQN learns to nearly constantly play the predominant action, but does not learn proper avoidance strategies, leading to a reward of $\approx$ 25 (i.e successfully crossing the road 25 times). t-DQN on the other hand not only learns faster to repeatedly play the predominant action, but also learns proper avoidance strategies by learning to anticipate *when* a new decision has to be made, resulting in an average reward close to the best possible reward of 34. Here, DAR profits from the use of a coarse control, learning faster than vanilla DQN. However, similarly to vanilla DQN, DAR learns a policy that can only achieve a reward of 25, not learning to properly avoid cars.

[2]Results for Pong and BeamRider are given in Appendix H.



(a) MountainCar (b) QBert

Figure 8. Example States in which TempoRL makes new decisions. The agents are trained with a maximal skip-length of 10. (a) Example states in which TempoRL learned when to make new decisions in MountainCar, starting slightly to the right of the valley. (b) Example states of Qbert. To make it easier to see where QBert is in the images we highlight him as a red square and indicate the taken trajectory with a blue arrow.

5. Analysis of TempoRL Policies

To analyze TempoRL policies and the decisions *when* to act we selected trained agents and evaluated their policies on the environments they were trained on. Videos for all the behaviours we describe here are part of the supplementary[3]. In the tabular case we plot the key-states for an agent (see Figure 4) that can skip at most 7 steps ahead. On the Cliff environment TempoRL learns to make a decision in the starting state, once it has cleared the cliff, once before reaching the other side (since it can not skip more than 7 states) and once to go down into the goal. Similar observations can be made for all grid-worlds. This shows that in this setting our TempoRL agents are capable of skipping over unimportant states and learned *when* they are required to perform novel actions.

[3]Available at `github.com/automl/TempoRL`

Key-states in which TEMPORL decides to take new actions in the featurized MountainCar environment are shown in Figure 8a. Starting slightly to the right of the valley, the agent learns to gain momentum by making use of skips, repeating the left action[4]. As soon as TEMPORL considers the run-up to be sufficient to clear the hill on the right, it switches the action direction. From this point on TEMPORL sticks with this action and always selects the largest available skip-length (i.e. 10). Still, TEMPORL has to make many intermediate decisions, as the agent is limited by the maximal skip-length.

Finally, we evaluated TEMPORL's skipping behaviour on Qbert. An example of key states in which TEMPORL decides to make new decisions are given in Figure 8b. Our TEMPORL agent learns to use large skip-values to reach the bottom of the left column, lighting up all platforms in between. After that the agent makes use of large skips to light up the second diagonal of platforms. Having lit up a large portion of the platform, TEMPORL starts to make fewer uses of skips. This behaviour is best observed in the video provided in the supplementary. Also, note that we included all trained networks in our supplementary such that readers can load the networks to study their behaviour.

This analysis confirms that TEMPORL is capable of not only reacting to states but also learning to anticipate *when* a switch to a different action becomes necessary. Thus, besides the benefit of improved training speed through better guided exploration, TEMPORL improves the interpretability of learned policies.

6. Conclusion

We introduced skip-connections into the existing MDP formulation to propagate information about future rewards much faster by repeating the same action several times. Based on skip-MDPs, we presented a learning mechanism that makes use of existing and well understood learning methods. We demonstrated that our new method, TEMPORL is capable of learning not only how to act in a state, but also *when* a new action has to be taken, without the need for prior knowledge. We empirically evaluated our method using tabular and deep function approximation and empirically evaluated the learning behaviour in an adversarial setting. We demonstrated that the improved learning speed not only comes from the ability of repeating actions but that the ability to learn which repetitions are helpful provided the basis of learning *when* to act. For both tabular and deep RL we demonstrated the high effectiveness of our approach and showed that even in environments requiring mostly fine-control our approach performs well. Further, we evaluated the influence of exploration strategies, architectural choices and maximum skip-values of our method and showed it to be robust.

As pointed out by Huang et al. (2019), observations might be costly. In such cases, we could make use of TEMPORL to learn how to behave and when new actions need to be taken; when using the learned policies, we could use the learned skip behaviour to only observe after having executed the longest skips possible. All in all, we believe that TEMPORL opens up new avenues for RL methods to be more sample efficient and to learn complex behaviours. As future work, we plan to study distributional TEMPORL as well as how to employ different exploration policies when learning the skip policies and behaviour policies.

[4]Note, in the particular example given in Figure 8a the agent first performs the left action twice, each for one time-step before it recognizes that it is gaining momentum and it can make use of large skips.

Acknowledgements

The authors acknowledge support by the state of Baden-Württemberg through bwHPC, André Biedenkapp, Raghu Rajan and Frank Hutter by the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG as well as funding by the Robert Bosch GmbH, and Marius Lindauer by the DFG through LI 2801/4-1. The authors would like to thank Will Dabney for providing valuable initial feedback as well as Fabio Ferreira and Steven Adriaensen for feedback on the first draft of the paper.

References

Bacon, P., Harb, J., and Precup, D. The option-critic architecture. In S.Singh and Markovitch, S. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'17)*. AAAI Press, 2017.

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent tool use from multi-agent autocurricula. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, 2020. Published online: iclr.cc.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279, 2013.

Biedenkapp, A., Bozkurt, H. F., Eimer, T., Hutter, F., and Lindauer, M. Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In Lang, J., Giacomo, G. D., Dilkina, B., and Milano, M. (eds.), *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*, pp. 427–434, June 2020.

Braylan, A., Hollenbeck, M., Meyerson, E., and Miikkulainen, R. Frame skip is a powerful parameter for learn-

ing to play atari. In *Proceedings of the Workshops at Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*, 2015.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *arXiv:1606.01540 [cs.LG]*, 2016.

Chaganty, A., Gaur, P., and Ravindran, B. Learning in a small world. In van der Hoek, W., Padgham, L., Conitzer, V., and Winikoff, M. (eds.), *International Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2012*, pp. 391–397. IFAAMAS, 2012.

Chrpa, L. and Vallati, M. Improving domain-independent planning via critical section macro-operators. In Hentenryck, P. V. and Zhou, Z. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'19)*, pp. 7546–7553. AAAI Press, 2019.

Dabney, W., Ostrovski, G., and Barreto, A. Temporally-extended ϵ-greedy exploration. *arXiv:2006.01782 [cs.LG]*, 2020.

Doya, K. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

Eysenbach, B., Salakhutdinov, R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pp. 15220–15231, 2019.

Hallak, A., Castro, D. D., and Mannor, S. Contextual markov decision processes. *arXiv:1502.02259 [stat.ML]*, 2015.

Harb, J., Bacon, P., Klissarov, M., and Precup, D. When waiting is not an option: Learning options with a deliberation cost. In McIlraith, S. and Weinberger, K. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'18)*, pp. 3165–3172. AAAI Press, 2018.

Harutyunyan, A., Vrancx, P., Bacon, P., Precup, D., and Nowé, A. Learning with options that terminate off-policy. In McIlraith, S. and Weinberger, K. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'18)*, pp. 3173–3182. AAAI Press, 2018.

Harutyunyan, A., Dabney, W., Borsa, D., Heess, N., Munos, R., and Precup, D. The termination critic. In Chaudhuri, K. and Sugiyama, M. (eds.), *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 89 of *Proceedings of Machine Learning Research*, pp. 2231–2240. PMLR, 2019.

Huang, Y., Kavitha, V., and Zhu, Q. Continuous-time markov decision processes with controlled observations. In *Proceedings of the 57th Annual Allerton Conference on Communication, Control, and Computing*, pp. 32–39. IEEE, 2019.

Khan, A., Feng, J., Liu, S., and Asghar, M. Z. Optimal skipping rates: training agents with fine-grained control using deep reinforcement learning. *Journal of Robotics*, 2019, 2019.

Khetarpal, K. and Precup, D. Learning options with interest functions. In Hentenryck, P. V. and Zhou, Z. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'19)*, pp. 9955–9956. AAAI Press, 2019.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*, 2015. Published online: iclr.cc.

Lakshminarayanan, A. S., Sharma, S., and Ravindran, B. Dynamic action repetition for deep reinforcement learning. In S.Singh and Markovitch, S. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'17)*, pp. 2133–2139. AAAI Press, 2017.

Lee, J., Lee, B., and Kim, K. Reinforcement learning for control with multiple frequencies. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, volume 33, 2020.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'16)*, 2016. Published online: iclr.cc.

Mankowitz, D. J., Mann, T. A., Bacon, P., Precup, D., and Mannor, S. Learning robust options. In McIlraith, S. and Weinberger, K. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'18)*, pp. 6409–6416. AAAI Press, 2018.

Metelli, A., Mazzolini, F., Bisi, L., Sabbioni, L., and Restelli, M. Control frequency adaptation via action persistence in batch reinforcement learning. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. Proceedings of Machine Learning Research, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control

through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Modi, A., Jiang, N., Singh, S. P., and Tewari, A. Markov decision processes with continuous side information. In *Algorithmic Learning Theory (ALT'18)*, volume 83 of *Proceedings of Machine Learning Research*, pp. 597–618. PMLR, 2018.

Moore, A. W. *Efficient memory-based learning for robot control*. PhD thesis, Trinity Hall, University of Cambridge, Cambridge, 1990.

Nasiriany, S., Pong, V., Lin, S., and Levine, S. Planning with goal-conditioned policies. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pp. 14814–14825. 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pp. 8024–8035, 2019.

Precup, D., Sutton, R. S., and Singh, S. P. Theoretical results on reinforcement learning with temporally abstract options. In *Proceedings of the 10th European Conference on Machine Learning (ECML)'98*, pp. 382–393, 1998.

Schoknecht, R. and Riedmiller, M. A. Speeding-up reinforcement learning with multi-step actions. In Dorronsoro, J. R. (ed.), *Proceedings of the International Conference on Artificial Neural Networks (ICANN'02)*, volume 2415 of *Lecture Notes in Computer Science*, pp. 813–818. Springer, 2002.

Schoknecht, R. and Riedmiller, M. A. Reinforcement learning on explicitly specified time scales. *Neural Computing and Applications*, 12(2):61–80, 2003.

Sharma, S., Lakshminarayanan, A. S., and Ravindran, B. Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

Stolle, M. and Precup, D. Learning options in reinforcement learning. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation SARA'02*, volume 2371 of *Lecture Notes in Computer Science*, pp. 212–223. Springer, 2002.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Tiganj, Z., Shankar, K. H., and Howard, M. W. Scale invariant value computation for reinforcement learning in continuous time. In *Proceedings of the AAAI Spring Symposia'17*, 2017.

Vallati, M., Chrpa, L., and Serina, I. MEvo: a framework for effective macro sets evolution. *Journal of Experimental & Theoretical Artificial Intelligence*, 0(0):1–19, 2019.

van Hasselt, H. Double q-learning. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A. (eds.), *Proceedings of the 23rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'10)*, pp. 2613–2621, 2010.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In Schuurmans, D. and Wellman, M. (eds.), *Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16)*, pp. 2094–2100. AAAI Press, 2016.

Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., and Agapiou, J. Strategic attentive writer for learning macro-actions. In Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16)*, pp. 3486–3494, 2016.

CHAPTER 12

Self-Paced Context Evaluation for Contextual Reinforcement Learning

The content of this chapter has been published as:

> T. Eimer, A. Biedenkapp, F. Hutter, and M. Lindauer (2021a). "Self-Paced Context Evaluation for Contextual Reinforcement Learning". In: *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 2948–2958.

Project Idea. The idea of using curriculum learning was proposed by Marius Lindauer and refined by Theresa Eimer to self-paced learning. André Biedenkapp proposed to use the value estimates of value-based RL agents to control the self-paced element. The concrete self-paced mechanism, based on value estimates was devised by Theresa Eimer with advice by Marius Lindauer. André Biedenkapp proposed the motivating example of cart-pole with varying pole lengths.

Implementation and experimentation. Implementation and experimentation were led by Theresa Eimer. André Biedenkapp aided in debugging and evaluated baselines on a subset of the considered environments.

Paper writing. An initial draft of the paper was written by Theresa Eimer. André Biedenkapp revised and edited the initial draft. Frank Hutter and Marius Lindauer revised the final paper version. The paper was to a large extent written by Theresa Eimer and André Biedenkapp.

Self-Paced Context Evaluation for Contextual Reinforcement Learning

Theresa Eimer [1] **André Biedenkapp** [2] **Frank Hutter** [2 3] **Marius Lindauer** [1]

Abstract

Reinforcement learning (RL) has made a lot of advances for solving a single problem in a given environment; but learning policies that generalize to unseen variations of a problem remains challenging. To improve sample efficiency for learning on such *instances* of a problem domain, we present *Self-Paced Context Evaluation (*SPACE*)*. Based on self-paced learning, SPACE automatically generates instance curricula online with little computational overhead. To this end, SPACE leverages information contained in state values during training to accelerate and improve training performance as well as generalization capabilities to new instances from the same problem domain. Nevertheless, SPACE is independent of the problem domain at hand and can be applied on top of any RL agent with state-value function approximation. We demonstrate SPACE's ability to speed up learning of different value-based RL agents on two environments, showing better generalization capabilities and up to $10\times$ faster learning compared to naive approaches such as round robin or SPDRL, as the closest state-of-the-art approach.

1. Introduction

Although Reinforcement Learning (RL) has performed impressively in settings like continuous control (Lillicrap et al., 2016), robotics (OpenAI et al., 2019) and game playing (Silver et al., 2016; Vinyals et al., 2019), their applicability is often very limited. RL training on a given task takes a lot of training samples, but the skills acquired do not necessarily transfer to similar tasks as they do for humans. An agent that is able to generalize across variations of a task, however, can be applied more flexibly and has a lower chance of succeeding when presented with unseen inputs. Therefore improving generalization means improving sample efficiency and robustness to unknown situations. We view these as important qualities for real-world RL applications.

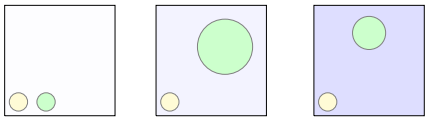


Figure 1: Example instances of the contextual PointMass environment. The agent's yellow starting point, the green goal and floor friction (indicated by shading) are part of the context and vary between instances.

Curriculum learning (Bengio et al., 2009) aims to bridge the gap between agent and human transfer capabilities by training an agent the same way a human would learn: transferring experience from easy to hard variations of the same task. It has been shown that generating such *instances* with increasing difficulty to form a training curriculum can improve training as well generalization performance (Dendorfer et al., 2020; Matiisen et al., 2017; Zhang et al., 2020). As information about instance difficulty is often not readily available, many approaches utilize the agent's progress markers, such as evaluation performance, confidence in its policy or its value function to minimize the need for domain knowledge (Wang et al., 2019; Klink et al., 2020). Because the progression is dictated by the agent's learning progress, this is called Self-Paced Learning (Kumar et al., 2010).

Instances in a curriculum can vary from the core task in different aspects, such as varying goals or movement speeds (see Fig. 1). While only selecting different goals states as instances is common for curriculum learning methods (Dendorfer et al., 2020; Zhang et al., 2020), changing transition dynamics are important considerations regarding the robustness of a policy. A dynamic change in robotics could for example be caused by a broken joint that the agent now has to adapt to. To allow these changes in the transition dynamics, in addition to goal changes in the instances, we consider *contextual* RL instead.

Our contributions are as follows:

1. We propose SPACE, a new self-paced learning algorithm, to automatically generate instance curricula in a

[1]Information Processing Institute (tnt), Leibniz University Hannover, Germany [2]Department of Computer Science, University of Freiburg, Germany [3]Bosch Center for Artificial Intelligence, Renningen, Germany. Correspondence to: Theresa Eimer <eimer@tnt.uni-hannover.de>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

general contextual RL setting, without any knowledge about instance difficulty being required and with access to only a limited set of instances (see Section 4).

2. We show the convergence behavior of SPACE to be at least as good as round robin (see Section 4.2).

3. We demonstrate that SPACE is capable of outperforming a round robin baseline (Speck et al., 2020) as well as similar self-paced methods (see Section 5).

2. Related Work

There are different approaches to increase generalization capability in RL. Their goals and scopes differ substantially, however. MAML (Finn et al., 2017) and related meta-RL methods pre-train an agent such that specializing on one of the training tasks is then very efficient. These take different approaches of aggregating and propagating the gradients in training and are complementary approaches to SPACE.

Domain randomization (DR; Tobin et al., 2017) on the other hand varies the task space. In essence, DR creates new instances of tasks in order to force the agent to adapt to alterations in its observations and policy. Other examples such as POET (Wang et al., 2019) and ADR (OpenAI et al., 2019) sample instances at random but order them by leveraging knowledge about the environment. Without prior knowledge of the target distribution, however, making appropriate changes is hard, resulting in either too little variation to facilitate generalization or deviating so much that the problem becomes too hard to learn. Other approaches utilize human expert knowledge to facilitate generalization performance, such as human-in-the-loop RL (Thomaz & Breazeal, 2006) or imitation learning (Hussein et al., 2017).

Curriculum learning (Bengio et al., 2009) uses expert knowledge to generate an ordering of training instances in such a way that knowledge can be transferred from hard to easy instances. There are different approaches for automatically generating such instance curricula, including learning how to generate training instances (Dendorfer et al., 2020; Such et al., 2020) similar to a teacher (Matiisen et al., 2017; Turchetta et al., 2020) or leveraging self-play as a form of curriculum generation (Sukhbaatar et al., 2018; da Silva et al., 2019). In most of these cases, some knowledge of the instance space is required in order to either define a measure of instance difficulty or how to generate new instances. While instance generation requires only little prior knowledge, a separate agent will need to learn to generate instances of appropriate difficulty, which increases the training overhead significantly.

Value Disagreement based Sampling (VDS; Zhang et al., 2020) on the other hand builds curricula for goal-directed RL. VDS uses the disagreement between different agents trained on the same instances to measure which training instance should be trained on next. Like its building block HER (Andrychowicz et al., 2017), VDS is only compatible with goal-directed off-policy RL.

One approach to order the training instances is explicitly using an agent's performance as an ordering criterion instead, called Self-Paced Learning. This can be done using the agent's value function as a substitute for actual episode evaluations. SPDRL (Klink et al., 2020) uses this idea to generate new instances uniquely suited to the agent's current training progress in order to progress towards specific hard instances. While this eliminates the need for a teacher, researchers instead need to know a priori which instances are considered the hard target instances and where the agent should start training in relation to them.

3. Contextual Reinforcement Learning

Before we describe SPACE, we discuss how we can extend the typical RL formulation to allow for the notion of instances. RL problems are generally modeled as Markov Decision Processes (MDPs), i.e., a 4-tuple $\mathcal{M} \coloneqq (S, A, T, R)$ consisting of a state space S, a set of actions A, a transition function $T : S \times A \to S$ and a reward function $R : S \times A \to \mathbb{R}$. This abstraction however, only allows to model a specific instantiation of a problem and does not allow to deviate from a single *fixed* instance.

An instance $i \in \mathcal{I}$ in a set of instances $\mathcal{I}$ could, e.g., determine a different goal position in a maze problem or different gravity conditions (i.e., moon instead of earth) for a navigation task. Information about the instance at hand is called its context c_i. This context can either directly encode information about the instance, e.g., the true goal coordinates, or the kind of robot that should be controlled.

In order to make use of context in our problem description, we consider contextual MDPs (cMDP; Hallak et al., 2015; Modi et al., 2018; Biedenkapp et al., 2020). A contextual MDP $\mathcal{M}_\mathcal{I}$ is a collection of MDPs $\mathcal{M}_\mathcal{I} \coloneqq \{\mathcal{M}_i\}_{i \in \mathcal{I}}$ with $\mathcal{M}_i \coloneqq (S, A, T_i, R_i)$. As the underlying problem stays the same, we assume the possible state and action spaces are consistent across all instances; however, the transition and reward functions are unique to each instance.[1]

An optimal policy π^* for such a cMDP optimizes the expected return over all instances $\mathcal{I}$ with discount factor γ:

$$\pi^* \in \arg\max_{\pi \in \Pi} \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \sum_{t}^{T} \gamma^t R_i(s_t, \pi(s_t)) \qquad (1)$$

As the reward depends on the given instance i, an agent solving a cMDP can leverage the context c_i along with

[1] In goal-directed RL, instances can also only vary the reward function and keep dynamics constant.

the current state $s_t \in S$ in order to differentiate between instances.

4. Self-Paced Context Evaluation

In order to generate a curriculum without any prior knowledge of the target domain, our *Self-Paced Context Evaluation (*SPACE*)* takes advantage of the information contained in an agent's state value predictions. By modelling $V^\pi(s_t, c_i)$, the agent learns to predict the expected reward from state s_t on instance i when following the current policy π. Therefore, we propose $V^\pi(s_0, c_i)$ as an estimate of the total expected reward given a starting state s_0.[2]

Definition 1. *The* performance improvement capacity (PIC) *of an instance is the difference in value estimation between point t and $t-1$, that is:*

$$d_t(i) = V_t^\pi(s_0, c_i) - V_{t-1}^\pi(s_0, c_i). \quad (2)$$

The intuition is, if the instance evaluation changes by a large amount, the agent has learned a lot about this instance in the last iteration and can potentially learn even more on it. Instances that are too easy or too hard will yield relatively small or no improvements. SPACE prefers instances on which it expects to make most learning progress. As most state-of-the-art RL algorithms use a value-based critic, each instance's PIC is easily computed during training.

Algorithm 1 summarizes the idea of SPACE. After some initialization in Lines 1-3, SPACE performs an update step for the current policy π and the value function V^π based on roll-outs on the current instance set $\mathcal{I}_{curr}$. In principle, any RL algorithm with a value-function estimate can be used, such as $\mathcal{Q}$-learning or policy search based on an actor-critic. In Lines 6-7, SPACE updates the average instance evaluation and the difference to the last iteration; note that this only considers the current set of instances $\mathcal{I}_{curr}$. In Lines 8-9, SPACE first checks whether the value function V_t^π changed ΔV_t^π by a factor $\eta < 1.0$ compared to the value function before the update. If the update led to an insignificant change of the value function, SPACE assumes that the learning sufficiently converged and we can add κ new instances to $\mathcal{I}_{curr}$. Starting in Line 10, SPACE determines which instances in $\mathcal{I}$ should be included in $\mathcal{I}_{curr}$. For each instance, SPACE first computes how much the value function changed, $d_t(i)$. The instances with the highest PIC regarding V^π are chosen as $\mathcal{I}_{curr}$ (Lines 12-13), assuming that it is easy to make progress on these instances right now. Note, we evaluate the influence of the η and κ hyperparameters on the learning behaviour of SPACE in our experiments.

[2]For simplicity's sake, we assume that an environment has a single starting state s_0 and we do not integrate over all possible starting states.

Algorithm 1: SPACE curriculum generation

Data: policy π, value function V, Instance set $\mathcal{I}$, threshold η, step size κ, #iterations T

1 $S, t := 0$
2 $V_0 := 0$
3 $\mathcal{I}_{curr} := \{i\}$ with i randomly sampled from $\mathcal{I}$
4 **for** $t = 1...T$ **do**
5 $\quad \pi, V_t^\pi := update(\pi, V_{t-1}^\pi, \mathcal{I}_{curr})$
6 $\quad V_t^\pi := \frac{1}{|\mathcal{I}_{curr}|} \sum_{i \in \mathcal{I}_{curr}} |V_t^\pi(s_0, c_i)|$
7 $\quad$ **if** $V_t^\pi \in [(1-\eta)V_{t-1}^\pi, (1+\eta)V_{t-1}^\pi]$ **then**
$\quad\quad$ `// Increase set size`
8 $\quad\quad S := S + \kappa$
$\quad$ `// Choose next instance set`
9 $\quad$ **forall** $i \in \mathcal{I}$ **do**
10 $\quad\quad d_t(i) := V_t^\pi(s_0, c_i) - V_{t-1}^\pi(s_0, c_i)$
11 $\quad \mathcal{I}_{curr} :=$ S instances with highest $d_t(i)$
12 $\quad t := t + 1$

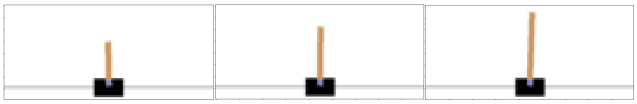


Figure 2: CartPole instances: short (s), medium (m) and long (l) balancing pole.

4.1. Exemplary Application of SPACE

As a motivating example, we consider the CartPole environment (Brockman et al., 2016) with three different pole lengths, see Figure 2. We use a small DQN (hyperparameters given in Appendix B) for this example with the pole length being given as an additional state feature. Although CartPole is generally considered as easy to solve, using poles of different length causes the DQN using a round robin curriculum to be unable to improve over time (see Figure 3). SPACE on the other hand is able to generate curricula that allow the DQN to learn how the cart has to be moved for the different poles and thus improve considerably to a mean performance of around 150 per episode compared to round robin's 25.

In Figure 4 we can see the main difference between the two methods. While the round robin agent trains on all three different variations one episode each, SPACE only chooses to train on the cart with the long pole twice before episode 40. Instead, the focus is on a single instance at a time, using either the short or medium pole and changing not every episode but trains on an instance for at least three consecutive episodes. This shows that the value function can provide guidance as to instance similarity, as we would expect that the short and medium sticks behave in a similar way, as well as difficulty, the long pole being the hardest to

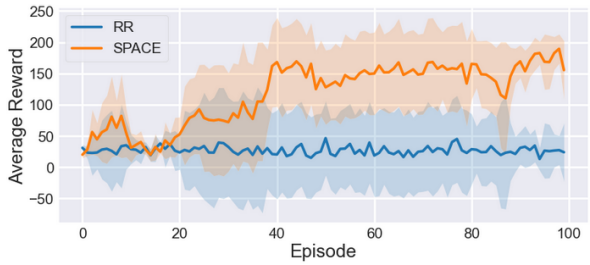


Figure 3: Performance (± std.) comparison of SPACE and default instance ordering on CartPole over 5 seeds each.

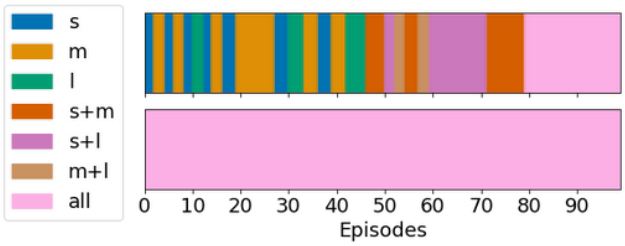


Figure 4: One exemplary run of SPACE (top) and round robin (bottom) curricula on CartPole.

control of the three. While the changes in the value function may not provide a completely stable curriculum, training on one instance for a flexible amount of episodes instead of one episode already has a big impact on overall performance. Furthermore, comparing the curriculum to the performance curve, focusing on only one instance at a time already leads to the agent performing considerably better on all of them. This validates the idea that there are underlying dynamics common to all three pole lengths which are important to learn and then refine according to the instance dynamics.

4.2. Convergence of SPACE

To discuss under which conditions SPACE will converge, we consider two cases.

Theorem 1. *Given a set of instances $\mathcal{I}$ that are sufficiently distinguishable by their context c_i as well as an instance of* SPACE *with $\eta > 0$, $\kappa \geq 1$ and an agent with value function V_t. If the value function estimation V_t^π converges in the limit to some value function V on each instance ($\forall i \in \mathcal{I}. \forall s \in S : \lim_{t\to\infty} V_t^\pi(s, c_i) = V(s, c_i)$) and globally ($\lim_{t\to\infty} V_t^\pi(s) = V(s)$),* SPACE *will eventually include all instances in the curriculum.*

Proof. Since for all $i \in \mathcal{I}$:

$$\lim_{t\to\infty} V_t^\pi(s_0, c_i) = V(s_0, c_i) \tag{3}$$

and therefore it follows that:

$$\lim_{t\to\infty} \Delta V_{t-1}^\pi = ||V_t^\pi(s_0, c_i)| - |V_t^\pi(s_0, c_i)|| \to 0 \tag{4}$$

Thus SPACE is guaranteed to include at least one other instance i' in the new curriculum $\mathcal{I}_{curr}$ at some point t. Now we assume that we are given any $\mathcal{I}' \subseteq \mathcal{I}$ with size $n < |\mathcal{I}|$. As $\forall i \in \mathcal{I}. \forall s \in S : \lim_{t\to\infty} V_t^\pi(s, c_i) = V(s, c_i)$ and Equation 3, convergence of V_t^π on the subset $\mathcal{I}'$ follows:

$$\forall i \in \mathcal{I}' : \lim_{t\to\infty} V^\pi(s_0, c_i) = V(s_0, c_i) \tag{5}$$

Therefore, as in the single instance case:

$$\lim_{t\to\infty} \Delta V_t^\pi \to 0 \tag{6}$$

and a new instance is added.

As the curriculum is guaranteed to be extended for any instance set of size $n = 1$ and $n \leq |\mathcal{I}|$, SPACE will eventually construct a curriculum using the whole instance set.

□

Corollary 1. *If* SPACE *covers all instances at some time point, it will be only slower than round robin by a constant factor in the worst case.*

Proof. Assume $\kappa = 1$ and that the learning agent requires $\mathcal{O}(K)$ steps to converge on a single instance.

If the agent is not able to transfer any of its gained knowledge between any of the tasks, SPACE will require to train an agent $\mathcal{O}(k)$ steps before growing the curriculum, where $k \leq K$, depending on η. SPACE will thus require $\mathcal{O}(|\mathcal{I}| \cdot k)$ steps to include all $|\mathcal{I}|$ instances in the curriculum. At this point, SPACE behaves as a round robin schedule does, i.e., iterating over each instance while training the agent. Therefore, even if the construction of a meaningful curriculum should have failed, SPACE can recover by falling back to a round robin scheme after $\mathcal{O}(|\mathcal{I}| \cdot k)$ steps. □

Corollary 2. *If the value function estimation converges to the* true *value function V^*,* SPACE *will also converge to the optimal policy.* □

Assume the worst case in which the value function estimate does not converge, but either oscillates or even diverges. This could happen if SPACE jumps between two disjoint instance sets $\mathcal{I}_1$ and $\mathcal{I}_2$ and the progress on $\mathcal{I}_1$ is lost by switching to $\mathcal{I}_2$ and vice versa.[3] Whenever we detect that learning is not progressing further and convergence is not achieved (i.e., $\Delta V_t^\pi \neq 0$ and $\mathcal{I}_{curr} \neq \mathcal{I}$), SPACE could simply increase η. As this hyperparameter controls how strict the convergence criterion is, increasing the value will allow for new instances to be added to the training set even though the original convergence criterion has not been met. The least value to which η should be set to guarantee an

[3]Note: Though theoretically possible, we have never observed this problem in practice.

increase of instances is $\frac{\Delta V_t^\pi + \epsilon}{V_{t-1}^\pi}$ for any $\epsilon > 0$ to eventually train on all instances.

Theorem 2. *If the value function estimate is not guaranteed to converge (e.g., in deep reinforcement learning),* SPACE *can still recover a round robin scheme by increasing the threshold η if needed.*

Proof. If at any point t, $\Delta V_t^\pi \neq 0$ and $\mathcal{I}_{curr} \neq \mathcal{I}$, we apply the method described above and set $\eta = \frac{\Delta V_t^\pi + \epsilon}{V_{t-1}^\pi}$. Then the condition to increase the instance set size is $\Delta V_t^\pi < \eta \cdot V_{t-1}^\pi \rightarrow \Delta V_t^\pi < \frac{\Delta V_t^\pi + \epsilon}{V_{t-1}^\pi} \cdot V_{t-1}^\pi \rightarrow \Delta V_t^\pi < \Delta V_t^\pi + \epsilon$. Thus the instance set size is guaranteed to be increased. As this is true for any point in training, SPACE can still consider all instances at some point t^* and thus perform as well as round robin from t^* onward. □

5. Experiments

In this section we empirically evaluate SPACE on two different environments. The code for all experiments is available at https://github.com/automl/SPaCE. We first describe the experimental setup before comparing SPACE against a round robin (RR) training scheme and SPDRL (Klink et al., 2020) as a state-of-the-art self-paced RL baseline. Finally we evaluate the influence of SPACE's own hyperparameters and limitations.

5.1. Setup

We evaluated SPACE in settings that readily allow for context information to encode different instances, namely the *Ant* locomotion environment (Coumans & Bai, 2020), the gym-maze environment (Chan, 2019) and the *BallCatching* and contextual *PointMass* environments as used by Klink et al. (2020).

The task in Ant is to control a four legged ant robot towards a goal on a flat 2D surface as quick as possible. The context is given by the x- and y-coordinates of the goal. Goals that are close to the starting position are easier to reach and thus we expect them to be easier to learn and their policies to transfer to more difficult instances. Additionally, the context indicates if no or up to one of the four legs of the ant robot is immobilized, similar to (Seo et al., 2020). We uniformly sampled 200 instances which we split in equal sized, disjoint training and test sets (see Appendix A). The context of the maze environment (Chan, 2019), in which the task is to find the goal state, is given as the flattened 5x5 layout of the current instance. 100 training and test instances each were sampled using the given maze generator. The agent's goal in BallCatching is to direct a robot to catch a ball. The ball's distance from the robot as well as it goal position are given as context information. Training and test sets were each 100 instances large and uniformly sampled between our context bounds. In the PointMass environment (see Figure 1), an agent maneuvers a point mass through a goal in a two-dimensional space. The goal position, the width as well as the friction coefficient of the ground are given as context. We sampled 100 instances for training and testing, each for two different distributions. The first distribution is chosen to cover the space of possible instances, whereas the second distribution follows that of Klink et al. (2020) and focuses on an area around a particularly difficult instance (see Appendix A).

To be consistent and fair with respect to prior work, we trained a PPO agent (Schulman et al., 2017) for Ant and a TRPO agent for PointMass (Schulman et al., 2015) and base our curriculum generation on their value-based actors. For easier readability, all plots are smoothed over 10 steps. In order to monitor generalization progress over time, we evaluated the agent on all instances in the training and test set after each complete run through the training set. As the results on training and test sets were very similar, we only report the test performance. In all experiments we evaluated our agents over 10 random seeds. For hardware specifications and hyperparameters, please see Appendix B.

5.2. Baselines

In our experiments, we use three different baselines to compare SPACE's performance to.

Round Robin (RR) To be sure SPACE outperforms instances without an intentional ordering, we compare against round robin as a common default instance ordering. This means that the training instances are ordered in an arbitrary way and we simply iterate over them, playing one episode per instance. As the instance sets we use are generated randomly, this ordering is chosen at random as well.

SPDRL SPDRL (Klink et al., 2020) is a state-of-the-art self-paced learning method for contextual RL. This is notable as most curriculum learning methods are explicitly designed for goal-directed RL, which makes them unsuitable in our setting. Counter to SPACE and RR, SPDRL makes use of an instance distribution to continually sample new instances of a specific difficulty level. SPDRL uses this ability to generate new instances to focus on particularly difficult instances, while largely ignoring the remaining instance space. To this end, SPDRL requires additional domain knowledge, besides the context information, to determine which instances SPDRL should focus on. Therefore we provide SPDRL with the distribution of our training and test set to focus the learning on its center.

cSPACE With SPACE we opted for taking an agent's knowledge about the expected reward, i.e. value function, into account to determine the similarity and difficulty of

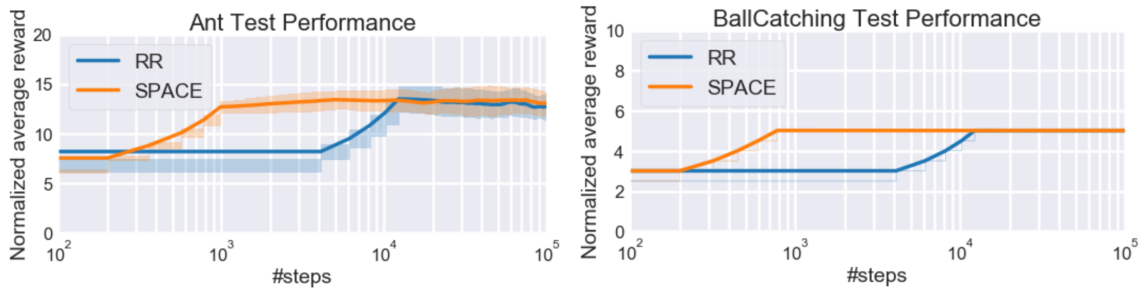


Figure 5: Mean reward ($\pm$ standard deviation) per episode over 10 runs on Ant (left) and BallCatching (right).

instances. However, as instances can be represented by their context, their similarity could also be quantified directly through their similarity in the context space. This form of similarity quantification is common in fields making use of techniques such as algorithm selection (Rice, 1976) and meta-learning (Brazdil et al., 2008). Such curricula order instances according to their context similarities. A successful application of this approach can be seen in Reverse Curriculum Generation for Reinforcement Learning (Florensa et al., 2017) where robot arm starting positions were ordered into a curriculum according to their similarity. In other words, instead of using an agent's performance evaluations as a basis for the curriculum generation, instances with contexts that are closest to the current curriculum context are added. SPACE's instance ordering criterion can easily be changed to compare context space distance instead of evaluations, yielding a variation we call *context* SPACE (cSPACE). More precisely, we replaced d as our instance selection criterion (see Algorithm 1 Line 10) with the Euclidean distance to the current instance set $\mathcal{I}_{curr}$. In such cases, cSPACE can suffer from the same problems as unsupervised learning. A priori it is not clear how to scale and weight the different context features without having any signal how the features will affect the difficulty of instances and how good the resulting curriculum will be. In contrast to SPACE, we deem this a potential challenge in applying cSPACE. For this reason, we recommend SPACE as the default approach whenever state evaluations are available.

5.3. Does the Instance Order Matter?

We first compare SPACE to a baseline round robin (RR) agent on the Ant and BallCatching environments, to determine if SPACE can find a curriculum that outperforms a random ordering. In Figure 5, both agents reach the same final performance in each environment, but the agent trained via SPACE learns considerably faster. It only requires 10^3 steps to reach a reward of around 11 in Ant whereas RR requires roughly $10\times$ as many steps to train an agent to reach the same reward. The results for BallCatching are similar, with SPACE again being faster to reach the final

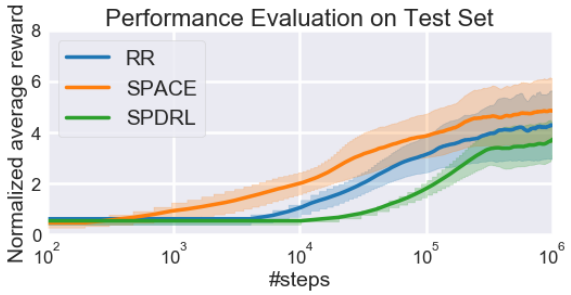


Figure 6: Mean reward per episode on a test set of uniformly sampled instances for PointMass.

performance a factor of at least 10. We further compare both methods on the PointMass environment when training on an instance set that was uniformly sampled from the space of possible instances (see Figure 6).

Here, the agent trained with SPACE is only roughly twice as fast, but it substantially outperforms round robin in terms of final performance. As the RR baseline does not care about the order in which instances are presented to the agent, we conclude that a more structured learning approach is needed. From SPACE's performance we can conclude that a curriculum, learned in a self-paced fashion can help improve both training performance and generalization. The experiments in the following sections further confirm this finding.

5.4. Comparing SPACE and SPDRL

We further compared SPACE to SPDRL (Klink et al., 2020) on the PointMass environment in order to demonstrate the difference between SPACE and an other self-paced learning method. We used the same implementation and hyperparameters as in (Klink et al., 2020) for SPDRL. The test performance of the agents can be seen in Figure 6.

As SPDRL was developed to train an agent to solve specific hard instances in the PointMass environment, it clearly falls short when it comes to covering the whole instance

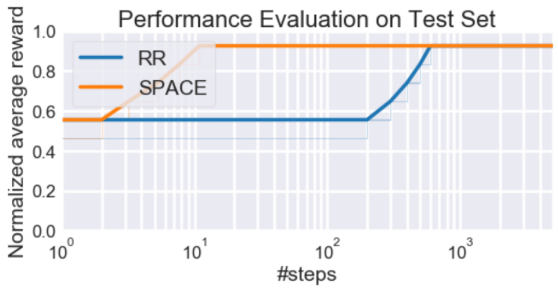


Figure 7: SPACE and RR of a set of mazes over 10 runs ($\pm$ standard deviation)

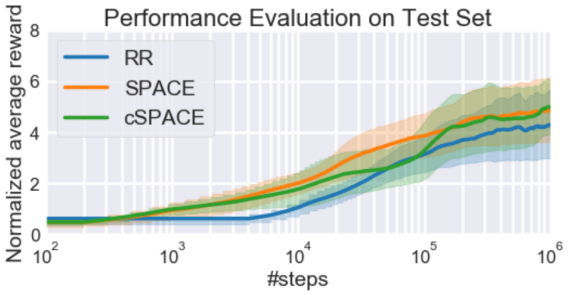


Figure 8: Mean reward on contextual PointMass with additional cSPACE results.

space. The agent trained via SPDRL learns much slower and achieves a worse final performance than an agent trained via RR. Perhaps unsurprisingly, this shows that targeting learning on hard instances does not imply the same agent can achieve good generalization performance on all instances.

5.5. How Well does SPACE Handle Complex Contexts?

While our benchmarks above are common meta-RL problem settings with different complexities, their contexts are given in a rather simple form, i.e., a short context vector directly describes the goal and environment dynamics. The agent can therefore make a direct connection between the changes between instances and the different context descriptions. This may not always be the case with context possibly being given within the observation, e.g., as part of an image.

In order to confirm that such a context description still enables SPACE to select the appropriate next instance, we use a set of 100 5x5 mazes (Chan, 2019) for our agent to generalize over. The observation is the agent's current position while the context is given by the flattened maze layout.

This context is much more complex than the previously used ones by having a structure that has been flattened and its components do not directly correlate to an increase in difficulty. Furthermore, many of the components of the context may not change from instance to instance even though the layout, and therefore the required policy, will.

Figure 7 shows that while the context information for this task is much more complex than previously seen, SPACE still outperforms the round robin agent in a similar way than it does to for Ant and BallCatching. The round robin agent needs several hundred episodes to solve all mazes while SPACE is able to generalize from just 10 episodes. While the context complexity increases in this case, the value function is still able to differentiate between them enough to allow a distinction between different instances. Therefore we expect that the representation of the context is not a major concern for the performance of SPACE.

5.6. Can SPACE Be Applied Without a Value Function?

The PointMass environment has three different context features for which we can easily use the context space distance to construct a curriculum. SPACE and cSPACE perform similarly on PointMass (Figure 8) in terms of learning speed and overall performance, both reaching the same performance at the same speed, with SPACE learning faster on average between 10^4 and 10^5. This makes both SPACE variations a better choice than round robin.

Both cSPACE and SPACE are also consistent in the curricula they find. We measured this by comparing the frequency with which each instance was used in the training set compared to the weighted frequency which gives higher rank to instances chosen at earlier iterations (1 for the instance chosen first down to $\frac{1}{|Z|}$ for the instance chosen last). For cSPACE, both the frequency and weighted frequency stayed the same while for SPACE only the four least used instances differed in order between the two.

We also compared the mean instance distance between curriculum iterations to see which method allows for smoother transitions between tasks. Smooth transitions correlate to a handcrafted curriculum where instances are close together in the context space, making the curriculum easier to learn from a human perspective. SPACE moves the instance set around 4.7% each curriculum iteration while cSPACE moves by around 5.6%. The maximum induced change is 10.1% for SPACE and 13.3% for cSPACE approach.

As we can see from these comparisons, using the information contained in an agent's value function to construct a curriculum is very similar to using the context space distance. It needs to be said, however, that in PointMass the context reflects the environment dynamics in a very direct way, being made up of the x- and y-positions of the goal to reach and the friction coefficient. Therefore we would expect cSPACE to perform very well on such environments. The fact that the default SPACE setting performs similarly indicates that the value function contains the information necessary to

Table 1: Mean reward $\pm$ standard deviation for different hyperparameter values on PointMass after 10^6 steps.

	η			
κ	5%	10%	20%	40%
1	5.1 ± 0.7	4.8 ± 1.2	4.7 ± 1.2	5.2 ± 0.7
4	5.5 ± 0.5	5.2 ± 0.7	4.3 ± 1.2	4.4 ± 1.0
16	4.6 ± 1.1	3.7 ± 1.1	5.1 ± 1.2	4.8 ± 1.0
32	4.5 ± 1.1	4.6 ± 1.1	4.7 ± 1.3	5.0 ± 1.2

order instances into a curriculum of similar quality. As not all environments may have such simple changes between instances, we expect that cSPACE has limitations on those kinds of environments while we can expect the value-based SPACE variation to continue constructing high quality curricula even in that case.

5.7. How Robust is SPACE wrt its Hyperparameters?

SPACE comes with two hyperparameters, the performance threshold for curriculum interactions η and the instance increment κ. These hyperparameters interact with each other to make SPACE comparatively stable across different hyperparameter values (as seen in Figure 1).

By varying η for a given value of κ, we alter the degree of stability the agent's value estimates have to reach between training episodes. Depending on the problem at hand, the value estimates may never be perfectly stable, therefore a very low value for η may prevent the training set from expanding. On the other hand, a very large value will move SPACE closer to round robin. Thus we view η as the more important hyperparameter of the two.

Our study shows very little performance differences for different values of κ and η. In part, this is because PointMass instances are not too difficult in the mean, therefore adding many at once does not heavily disturb learning. Larger performance thresholds η are not an issue for this reason. A value of 5% for η seems quite low, but as the instances are relatively easy, the agent can still converge enough very quickly. Different values for κ show similar results here. We expect this hyperparameter to be more important in very diverse settings with large gaps between instances. We can see the effect if we multiply the size of our training set instead of adding instances (see Figure 9). In this case, there is a visible slowdown, supporting that κ has a big influence on training performance.

From these results, we believe that it is reasonable to recommend keeping $\kappa = 1$ for most applications. It can yield more fine-grained curricula which will be important on diverse instance sets and will likely only impact training on very large instance sets. For η, using a low value such as 5% should ensure that the agent will not be overwhelmed with new instances if it takes more than one curriculum iteration to learn from the current training set.

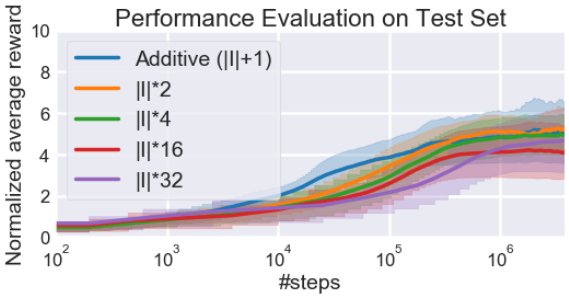


Figure 9: Mean reward per episode on a test set with fast rising instance set size (i.e. varying κ) and fixed $\eta = 10\%$.

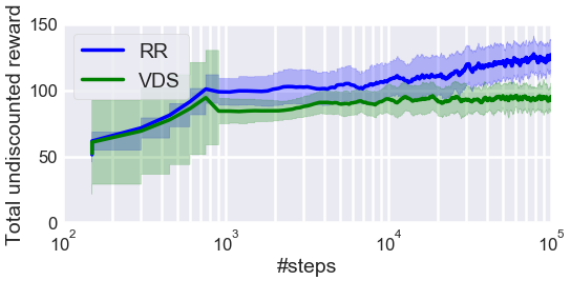


Figure 10: Total undiscounted reward of VDS and its RR baseline on AntGoal.

5.8. Can Goal-Directed RL Achieve Similar Results?

Goal-directed RL and contextual RL are closely related flavours of RL. The main difference between the two is that in goal-directed RL, the context is restricted to only contain information about a desired goal-location. Thereby the context indicates a (final) state in which the agent should reach. Policies learning with goals as context information thus can learn the value of executing an action in a certain state for reaching the desired goal. Crucially however, in this setting transition dynamics are assumed to never change for different contexts.

Contextual RL subsumes goal-directed RL by also allowing the environment dynamics for the same goal to vary. Both environments we evaluate on exemplify this. In contextual PointMass, we specify both a goal and the friction coefficient to describe an instance. Even if the goal is kept the same, however, the friction is a deciding factor in the amount of force that is necessary to reach the goal correctly. So while the force direction is the same for instances with the same goal, the required amount of force depends on the friction and therefore the agent needs both information to learn to generalize across different instances.

On AntGoal, we can see how this looks in practice. VDS (Zhang et al., 2020) is a recent state-of-the-art method for goal-directed curriculum construction based on HER (Andrychowicz et al., 2017). As a result, it can take only the ant's goal into consideration when selecting the next instance and crucially misses that in different instances the ant has different defects in some of its joints. As a result, the method conflates all instances with the same goals and fails to actually learn how to act on any of them.

We used the implementation and baseline of Zhang et al. (2020) to demonstrate that while goal-directed curriculum generation approaches seem similar to SPaCE, our problem setting is out of scope for them (see Figure 10). Their RR baseline has a different learning curve as ours as the algorithm used is different, but it clearly is able to improve over time. As VDS uses goals to describe the necessary behaviour, it cannot do the same. Therefore, curriculum learning methods for contextual RL and goal-directed RL have different scopes and cannot be compared fairly in the contextual RL setting.

6. Limitations

Even though SPACE performed very well on the benchmarks used in this paper, there are several limitations of SPACE to be considered. The first one is that the problem and the instance set both need to support curriculum learning to some degree. For the problem itself this means that the policy to solve it is influenced by context to a large degree, but that there is an underlying structure that can be exploited using a curriculum. The instance set then needs to be large enough to actually give SPACE the opportunity to do so. In settings with too little or very large amounts of instances, SPACE becomes less efficient (see Appendix D).

Furthermore, if the instance set is very homogeneous, similar to the specific instance SPDRL uses on PointMass (see Appendix C), using different instances for training might not make a difference. Conversely, if the instance set is heterogeneous, preliminary experiments showed that SPACE requires a larger amount of instances to speed up the learning. Thus not every problem is suited for curriculum learning.

Lastly, SPACE is constructed to work for discrete instance spaces only, where the instance ordering is essential for learning efficiency. We stress the fact that SPACE is designed for use cases with only a few instance examples. In settings with instance generators or a lot of domain knowledge available, it is likely better to exploit them which SPACE is not designed for.

7. Conclusion

Self-Paced Context Evaluation (SPACE) provides an adaptive curriculum learning method for problem settings constrained to a fixed set of training instances. Thereby we facilitate generalization in practical applications of RL. We demonstrated that the order of instances on which agents learn their behaviour policies indeed is important and can produce a better learning efficiency. In addition, SPACE outperformed a simple round robin baseline as well as more specialized curriculum learning methods requiring access to unlimited instance generators to perform well. Finally we evaluated the influence of SPACE's own hyperparameters and showed that they are robust on the chosen environments.

Future research could address how to derive performance expectations for practical applications of RL with a limited amount of instances with respect to the amount of information available. Furthermore, we might be able to use value estimation to further improve training efficiency for example by clustering instances of similar difficulty and limiting the amount of training on very easy ones to a minimum. Another important factor for contextual RL in general is catastrophic forgetting (see Appendix F), which is not yet sufficiently understood, especially in the continuous context spaces we applied SPACE to.

8. Acknowledgements

Theresa Eimer and Marius Lindauer acknowledge funding by the German Research Foundation (DFG) under LI 2801/4-1. All authors acknowledge funding by the Robert Bosch GmbH.

References

Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. Hindsight experience replay. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*, pp. 5048–5058, 2017.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In Bottou, L. and Littman, M. (eds.), *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*, pp. 41–48. Omnipress, 2009.

Biedenkapp, A., Bozkurt, H. F., Eimer, T., Hutter, F., and Lindauer, M. Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In Lang, J., Giacomo, G. D., Dilkina, B., and Milano, M. (eds.), *Proceedings of the Twenty-fourth European Conference*

on Artificial Intelligence (ECAI'20), pp. 427–434, June 2020.

Brazdil, P., Giraud-Carrier, C., Soares, C., and Vilalta, R. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition, 2008.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Chan, M. gym-maze. `https://github.com/MattChanTK/gym-maze`, 2019.

Coumans, E. and Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2020.

da Silva, F. L., Costa, A. H. R., and Stone, P. Building self-play curricula online by playing with expert agents in adversarial games. In *8th Brazilian Conference on Intelligent Systems, BRACIS '19*, pp. 479–484, 2019.

Dendorfer, P., Osep, A., and Leal-Taixé, L. Goal-GAN: Multimodal trajectory prediction based on goal position estimation. In *Proceedings of the 15th Asian Conference on Computer Vision (ACCV'20)*, 2020.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML '17)*, volume 70, pp. 1126–1135, 2017.

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. Reverse curriculum generation for reinforcement learning. In *Proceedings of the 1st Conference on Robot Learning (CoRL'17)*, volume 78, pp. 482–495, 2017.

Hallak, A., Castro, D. D., and Mannor, S. Contextual markov decision processes. *arXiv:1502.02259 [stat.ML]*, 2015.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2):21:1–21:35, 2017.

Klink, P., D'Eramo, C., Peters, J., and Pajarinen, J. Self-paced deep reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS'20)*, 2020.

Kumar, M. P., Packer, B., and Koller, D. Self-paced learning for latent variable models. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A. (eds.), *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'10)*, pp. 1189–1197, 2010.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'16)*, 2016. Published online: `iclr.cc`.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher-student curriculum learning. *CoRR*, abs/1707.00183, 2017.

Modi, A., Jiang, N., Singh, S. P., and Tewari, A. Markov decision processes with continuous side information. In *Algorithmic Learning Theory (ALT'18)*, volume 83, pp. 597–618, 2018.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. Solving rubik's cube with a robot hand. *arXiv:1910.07113 [cs.LG]*, 2019.

Rice, J. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, volume 37, pp. 1889–1897. Omnipress, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv:1707.06347 [cs.LG]*, 2017.

Seo, Y., Lee, K., Gilaberte, I. C., Kurutach, T., Shin, J., and Abbeel, P. Trajectory-wise multiple choice learning for dynamics generalization in reinforcement learning. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS'20)*, 2020.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Speck, D., Biedenkapp, A., Hutter, F., Mattmüller, R., and Lindauer, M. Learning heuristic selection with dynamic algorithm configuration. In *Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL@ICAPS'20)*, October 2020.

Such, F. P., Rawal, A., Lehman, J., Stanley, K. O., and Clune, J. Generative teaching networks: Accelerating

neural architecture search by learning to generate synthetic training data. In III, H. D. and Singh, A. (eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML'20)*, volume 98. Proceedings of Machine Learning Research, 2020.

Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. Intrinsic motivation and automatic curricula via asymmetric self-play. In *6th International Conference on Learning Representations (ICLR '18)*, 2018.

Thomaz, A. L. and Breazeal, C. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI'06)*, pp. 1000–1006, 2006.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '17)*, pp. 23–30, 2017.

Turchetta, M., Kolobov, A., Shah, S., Krause, A., and Agarwal, A. Safe reinforcement learning via curriculum induction. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, (NeurIPS'20)*, 2020.

Vinyals, O., Babuschkin, I., Czarnecki, W., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J., Jaderberg, M., Vezhnevets, A., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T., Gülçehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, 2019.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. POET: open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'19*, 2019.

Zhang, Y., Abbeel, P., and Pinto, L. Automatic curriculum learning through value disagreement. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS'20)*, 2020.

CHAPTER 13

CARL: A Benchmark for Contextual and Adaptive Reinforcement Learning

The content of this chapter has been presented at a workshop as:

> C. Benjamins, T. Eimer, F. Schubert, A. Biedenkapp, B. Rosenhan, F. Hutter, and M. Lindauer (2021). "CARL: A Benchmark for Contextual and Adaptive Reinforcement Learning". In: *Workshop on Ecological Theory of Reinforcement Learning (EcoRL@NeurIPS'21)*.

Project Idea. The benchmark collection was proposed by Marius Lindauer. André Biedenkapp proposed to use physical properties as considered context features. Theresa Eimer and Carolin Benjamins jointly designed the context spaces for all considered environments. Theresa Eimer and Carolin Benjamins proposed most benchmarks. André Biedenkapp proposed the *vehicle racing* benchmark. Frederik Schubert proposed the ToadGAN benchmark.

Implementation and experimentation. Carolin Benjamins and Theresa Eimer jointly carried out implementation, dividing the work equally. André Biedenkapp implemented the vehicle racing benchmark. Frederik Schubert provided the ToadGAN benchmark. Carolin Benjamins and Theresa Eimer carried out experiments with support from Frederik Schubert for the ToadGAN benchmark.

Paper writing. Carolin Benjamins and Theresa Eimer wrote the initial draft. André Biedenkapp revised and edited the draft with support from Marius Lindauer. Frank Hutter and Bodo Rosenhahn provided feedback for the final revision. André Biedenkapp added the discussion of problems and challenges for learning general reinforcement learning agents with feedback from all authors. The paper was finalized by Carolin Benjamins and André Biedenkapp.

CARL: A Benchmark for Contextual and Adaptive Reinforcement Learning

Carolin Benjamins[*1], **Theresa Eimer**[*1], **Frederik Schubert**[1], **André Biedenkapp**[2], **Bodo Rosenhahn**[1], **Frank Hutter**[2 3] and **Marius Lindauer**[1]
[1]Leibniz University Hanover [2]University of Freiburg [3]Bosch Center for Artificial Intelligence
{benjamins,eimer}@tnt.uni-hannover.de

Abstract

While Reinforcement Learning has made great strides towards solving ever more complicated tasks, many algorithms are still brittle to even slight changes in their environment. This is a limiting factor for real-world applications of RL. Although the research community continuously aims at improving both robustness and generalization of RL algorithms, unfortunately it still lacks an open-source set of well-defined benchmark problems based on a consistent theoretical framework, which allows comparing different approaches in a fair, reliable and reproducible way. To fill this gap, we propose CARL, a collection of well-known RL environments extended to contextual RL problems to study generalization. We show the urgent need of such benchmarks by demonstrating that even simple toy environments become challenging for commonly used approaches if different contextual instances of this task have to be considered. Furthermore, CARL allows us to provide first evidence that disentangling representation learning of the states from the policy learning with the context facilitates better generalization. By providing variations of diverse benchmarks from classic control, physical simulations, games and a real-world application of RNA design, CARL will allow the community to derive many more such insights on a solid empirical foundation.

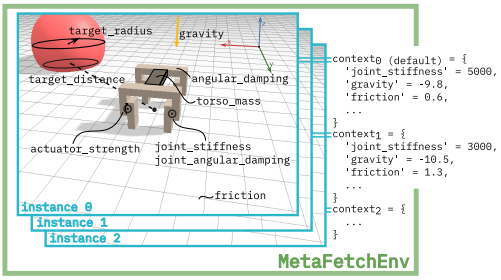
1 Introduction

Reinforcement Learning (RL) has driven progress in areas like game playing [53, 4], robot manipulation [27], traffic control [2], chemistry [65] and logistics [28]. At the same time, RL has shown little to no success in real-world deployment in important areas such as healthcare or autonomous driving. This is largely explained by the fact that modern RL agents are often not primarily designed for generalization, making them brittle when faced with even slight variations in their environment [61, 30, 33]. Since we cannot assume that RL agents will be able to observe all kinds of states and transitions for varying instances of tasks, these agents need to become more adaptable and robust.

To address this limitation, there is increased interest in Meta-RL approaches, aiming to improve learning across different tasks [15, 51, 12, 58, 32, 24, 38, 13]. The focus mostly lies on increasing the sample efficiency of agents, few-shot transfer of policies to new tasks or on solving harder tasks. Similarly, Robust-RL addresses generalization to smaller variations in the environment by ensuring a stable performance under task modelling errors or noisy observations [37, 40, 63]. While these directions are important in making RL more broadly and robustly applicable, with CARL we aim for providing the foundations of more general RL agents. Optimally, these agents should be capable of zero-shot transfer to prior unseen environments and changes in transition dynamics while interacting with an environment [62, 19, 60, 1, 55].

[*]Equal Contribution, Contact Author

Preprint. Under review.



(a) Example of a configurable CARL environment

(b) ProcGen Maze instance

Figure 1: (a) CARL makes the context defining the behavior of the environment visible and configurable. This way we can train for generalization over different instances (contexts) of the same environment. Here, we show all context features for Brax' Fetch [17] and sketch possible instantiations by setting the context features to different values. Fetch is embedded in the CARL environment controlling the instances. (b) A sampled ProcGen Maze instance.

Unfortunately, there is a lack of established benchmarks for studying the notion of generalization. In fact, we often observed that researchers employed hand-crafted modifications to commonly accepted tasks to enable benchmarking of Meta-RL. For example, multiple different modifications to the well known *CartPole* task, in which an agent needs to learn to balance a pole on top of a movable cart, have been used in different publications to show generalization abilities of agents [52, 23, 13]. In particular, different pole lengths are used to study whether a general agent can balance poles that it has not seen during training. However, the pole lengths or distributions over pole lengths vary in different publications, hindering comparisons and reproducibility. With CARL we provide such distributions to facilitate better comparability and reproducibility for further research.

Our goal is to address all these issues by proposing CARL, a benchmark library allowing to reliably and reproducibly study general RL agents. To this end, CARL has well defined distributions and bounds over the space of environments to generalize to and poses a low barrier of entry in terms of compute. To build on a sound theoretical foundation, we make use of the contextual Reinforcement Learning paradigm (cRL) [20] and build contextual extensions to environments from the literature including OpenAIs Gym [7] and the Brax physics engine [17]. The notion of context in the environment enables us to define a variety of tasks and distributions of tasks which an agent can encounter during training. Changes in tasks can be as simple as defining different goal states, more complex by changing the transition dynamics (as the changes in pole length for the CartPole environment mentioned above) or a combination thereof, leading to varying levels of difficulty. Most importantly and wherever possible, we base the notion of context on real-world physical properties, such as gravity, friction or mass of an object, see Figure 1a for an example of a contextually extended environment. Those properties are intuitive to understand and individually adjustable.

The proposed benchmark enables research on generalization capabilities of RL agents in cases where agents are explicitly or implicitly exposed to the context at hand but also in cases where the context is hidden and potentially has to be learned. In particular, we demonstrate the usefulness of our proposed CARL benchmark library by evaluating and discussing:

1. The influence of varying context and the importance of contexts in deep RL by increasing learning efficiency via available knowledge of task variations,
2. The generalization capability of trained agents to in-distribution environments,
3. The generalization capability of trained agents to out-of-distribution environments, and
4. The big next challenges for general RL which can be studied in principled way with CARL.

2 CARL's Theoretical Foundation: Contextual RL (cRL)

A basic MDP is a 4-tuple $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ consisting of a state space S, an action space A, a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. This abstraction, however, views reward and transition function as fixed, constraining the environment to a single instantiation

without room for variations we would potentially see in real-world applications. In the following we will refer to such a particular instantiation of an environment as an *instance*.

CARL's theoretical foundation is build upon contextual RL. It extends the MDP formulation of RL problems to allow for the notion of several instances. For example, an instance $i \sim \mathcal{I}$ sampled from a distribution of instances $\mathcal{I}$ could determine a different goal position in a maze problem (e.g., Figure 1b) or different gravity conditions (e.g., moon vs. earth) for an airborne navigation task. We refer to the information defining the instance at hand as the instance's context c_i. We note that c_i is sometimes known to the agent (e.g., a broken leg), sometimes measured with noise (e.g., friction of floor), or maybe even completely unobservable (e.g., mass of an external object). With CARL we provide a variety of such contexts that can influence an agent's learning and generalization capabilities. We further provide bounds and distributions of these contexts to facilitate better reproducibility and comparability for future research.

In order to incorporate context into the MDP definition, we use contextual MDPs (cMDPs) [20, 36, 5]. A contextual MDP $\mathcal{M}_\mathcal{I}$ is a collection of MDPs $\mathcal{M}_\mathcal{I} := \{\mathcal{M}_i\}_{i\sim\mathcal{I}}$ with $\mathcal{M}_i := (\mathcal{S}, \mathcal{A}, \mathcal{T}_i, \mathcal{R}_i)$. This formulation assumes a common state and action space as the underlying task stays the same; however, in each $\mathcal{M}_i$ an agent will potentially only be able to reach only a subset of states $S_i \subseteq S$. Transition and reward functions may vary between instances.

There are several different tasks of interest concerning cMDPs, all of which define an optimal policy π^* for a given cMDP in different ways. An example would be the focus on generalization performance where π^* maximizes the expected return across a test set $\mathcal{I}_{\text{Test}}$ drawn from the same distribution as the training set $\mathcal{I}_{\text{Train}}$ with discount factor γ [5]:

$$\pi^* \in \arg\max_{\pi\in\Pi} \frac{1}{|\mathcal{I}_{\text{Test}}|} \sum_{i\in\mathcal{I}_{\text{Test}}} \sum_{t}^{T} \gamma^t R_i(s_t, \pi(s_t)), \tag{1}$$

where T corresponds to the maximal episode length. In policy transfer, the focus is on the performance across a set of transfer instances specifically, which is often relatively small but rather heterogeneous. Here, the test instance set $\mathcal{I}_{\text{Test}}$ can largely differ from $\mathcal{I}_{\text{Train}}$, but the optimal policy would still maximize the mean reward across it just as above. On the other hand, only the final performance on a single, very hard instance i_H might be important and all other instances are only used to work towards that goal. That could be the case in, e.g., curriculum learning. In that case, we use the available training set $\mathcal{I}_{\text{Train}}$ to find a π^* such that:

$$\pi^* \in \arg\max_{\pi\in\Pi} \int_{\mathcal{I}_{\text{Train}}} P(i) \sum_{t}^{T} \gamma^t R_i(s_t, \pi(s_t)) di, \tag{2}$$

where the probability $P(i)$ of considering an instance i is skewed towards i_H over time.

cRL in CARL subsumes several other related formulations. For example, Goal-based RL [16] uses the same idea of conditioning the reward function of each task on its specific goal, but is more limited in scope as the environment dynamics stay static throughout. Block MDPs [11], on the other hand, focus on state representations for generalization. The task here is to learn a representation of the observable space of a family of environments that enables generalization across that family. Just as in cRL, reward functions and transition dynamics both may vary with the family, but the focus is shifted away from learning a policy towards learning a meaningful representation. While the original block MDP did not include specifics about how reward and transition functions differ within the environment family, contextual block MDPs provide the context as additional information [62]. As we have direct access to the context information on all CARL benchmarks, the base case provides context as in a cMDP. However, users are free to switch to a hidden context version that requires a representation learning as in a block MDP.

3 Related Work

Benchmarks for generalization exist in different sub-fields of RL, each with its own focus. MDP Playground [43] and bsuite [39] both contain small scale benchmarks intended to test specific qualities in RL algorithms (e.g., resistance to noise), both for the purpose of development and comparison between different algorithms. In contrast, the focus of CARL is less on assessing RL algorithms

Table 1: Comparison of the CARL Benchmark collection to related Benchmarks. Benchmarks are rated as true (✔), somewhat true (●) or false (✘) in each category.

Benchmark	Open Source	Explicit Context	Cheap Training[2]	Diverse Tasks	Varying $\mathcal{T}$&$\mathcal{R}$
MDPP [43]	●	●	✔	✔	✔
bsuite [39]	✔	✘	✔	✔	✘
ALE [31]	✔	✘	●	●	✘
ProcGen [10]	✔	✘	✘	✔	✘
Alchemy [57]	✔	●	✘	●	✔
Meta-world [61]	✘	✔	✘	●	●
MTEnv [54]	✘	✔	✘	●	●
Safety Gym [45]	✘	✘	✘	●	✔
TMA [47]	✔	●	✘	●	✔
MiniGrid [8]	✔	✘	✔	✔	●
NetHack [26]	✔	●	✘	✔	✘
MiniHack [49]	✔	●	✔	✔	✔
CARL (ours)	✔	✔	✔	✔	✔

against each other on fixed MDPs but in terms of their generalization capabilities to variations of MDPs. Benchmarks such as in MDP Playground and bsuite provide valuable feedback for researchers in development before they tackle more complex and opaque problems like the ones we provide.

In game simulations, the Arcade Learning Environment (ALE) has made an effort to include some challenges geared towards policy transfer and generalization in their "flavours" [31]. However, the bigger challenge in this field is ProcGen [10]. It contains several arcade-style games with procedurally generated level structures. In a similar way, Alchemy [57] also provides a procedurally generated benchmark. Even though it only contains a single task, this task is very complex compared to the games in ProcGen on their own. Both are challenging benchmarks that require generalization from state observations only. We believe that this approach is less valuable in many applications other than in games, because most often additional information is available. Additionally, while it is possible to specify levels with certain attributes in Alchemy, these procedurally generated benchmarks provide a far less fine-grained control over their context than the diverse set of benchmarks in CARL where users can directly specify their instances and control the similarity of their sampled contexts. CARL's flexibility allows for a better characterization of agents' generalization capabilities as well as the possibility of adding custom curricula for each environment.

Multi-task learning requires some amount of generalization, although here the focus is on accelerating the acquisition of skills on completely new tasks. For example, Meta-world [61] focuses on skill transfer in a few-shot setting, providing standardized test sets of different sizes. Its tasks are based on MuJoCo, however, which requires a paid license for large scale experiments, is comparatively much more expensive to run than the Brax physics simulator (used as part of CARL) and thus limits the accessibility of the benchmark. Meta-world is also integrated in MTEnv [54]. MTEnv provides a strong benchmark for multi-task learning as well as representation learning. CARL can accommodate multi-task learning as well, but the focus is on the multitude of context options available in each of our environments and therefore generalization across different transition dynamics.

There also exist more, related benchmarks in specialized subfields of RL. Safety Gym [45] is targeted towards developing and testing algorithms for risk-sensitive domains. Also, TeachMyAgent [47] is a benchmark for teacher-student based curriculum learning. Both are well suited to the needs of their communities, but also narrow in their scope. While CARL currently does not provide explicit contexts for either safety or curriculum learning, it could be extended to cover both domains and will be especially relevant for any curriculum learning algorithms not using the teacher-student framework.

Overall, CARL is the only benchmark library that is completely open-source, allows for fine-grained control of context on a diverse set of benchmarks and thus allows to study the next generation of general RL agents in a reliable and reproducible way. We summarize this comparison in Table 1.

[2]"Cheap Training" here refers the total runtime of one agent. This takes into account both to the computational cost of the environment itself as well as the number of training steps necessary to expect results.

4 The Role of Context in Deep RL and CARL

One important distinction that needs to be made in contextual RL concerns the ease of identifiability of context information. Here we broadly distinguish between explicit contexts, i.e. directly available information provided by the environment, and implicit contexts, i.e. abstract information hidden in the available state. While explicit contexts can directly be used by agents to infer the underlying transition dynamics, implicit contexts potentially need to be disentangled from the provided state. In particular, we argue that deep RL research commonly already makes use of the notion of contexts. This context however is only present in an implicit form in the state, thereby entangling representation learning capabilities with generalization capabilities of a policy.

For example in the ProcGen *maze* environment (see Figure 1b), an agent can observe the whole maze and is tasked with guiding a mouse from the bottom left corner to the cheese. The maze structure, texture of the walkable tiles and the location of the goal (i.e. cheese) are randomly generated for each new instance. Note that the wall texture never changes and that observations are only available as images. Thus, a capable RL agent could learn to directly extract the location of the mouse and cheese as well as classify which tiles are (not) walkable. This extracted information then allows the agent to perform contextual RL. In particular, Eimer et al. [13] showed that providing an agent with the coordinates of the agent and goal states as well as a flattened vector representation of a maze allows agents to make use of this context information to transfer behaviours between similar mazes.

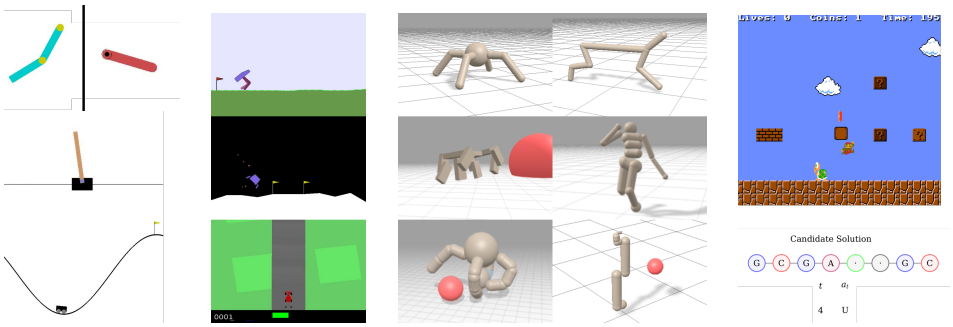
A similar argument can be made for more complex environments where a "level" might not be fully observable. For example, in the game Super Mario Bros., see Figure 2d, Mario needs to reach the goal on the right side of the screen while avoiding enemies. If an agent is made aware of the enemy types appearing in the level, through the use of context, this information can be used downstream in the policy net to learn appropriate offensive or defensive behaviour. Another direct context feature could be an indicator which special ability an agent can use, potentially leading to different behaviour when Mario picked up a power-up.

We argue that benchmarks using procedural content generation are more suitable for evaluating the representation learning capabilities of agents rather than their ability to generalize. In fact, the authors of ProcGen [10] used it to determine that the IMPALA-CNN [14] architecture is more capable than the Nature-CNN [34] architecture for their considered setup. Here, we propose benchmarks that provide a ground truth for the changes in underlying transition dynamics to study generalization while also containing more complex environments that can be used to study representation learning. Disentangling these two important tasks will enable researchers to target each more efficiently and ultimately facilitates the development of new RL algorithms targeted towards generalization. We use CARL to demonstrate that an agent making use of context information during training can learn to solve instances quicker and generalizes better than those that have to infer this information themselves (see Section 6.2). This gives additional evidence that disentangling learning of such contextual features from learning the behaviour policy can improve the generalization capabilities of RL agents (see Appendix D.1 for further discussion).

5 The CARL Benchmarks

In order to gain insight on how the context and its augmentation influences the agent's learning and behavior, we provide several benchmarks in CARL. As first benchmarks we include and contextually extend classic control and box2d environments from OpenAI Gym [7], Google Brax' walkers [17], a RNA folding environment [48] as well as Super Mario levels [3, 50]. See Figure 2 for an overview of included environments. Although each environment has different tasks, goals and mechanics, the behavior of the dynamics and the rewards is influenced by physical properties. A more detailed description of the environments is given in Appendix A. In the following we will discuss the properties of the CARL Benchmarks which are summarized in Figure 3.

State Space Most of our benchmarks have vector based state spaces that can either be extended to include the context information or not. The notable exceptions here are CARLVehicleRacing and CARLToadGAN, which exclusively use pixel-based observations. The size of the vector based spaces range from only two state variables in the CARLMountainCar environments to 299 for the CARLHumanoid environment.



(a) Classic Control [7] (b) 2D Physics Simulation [7] (c) Brax 3D Physics Simulation [17] (d) Super Mario [3, 50] & RNADesign [48]

Figure 2: CARL Environments; listed from top to bottom. (a) OpenAI Gym's [7] Acrobot and Pendulum, CartPole, MountainCar. (b) OpenAI Gym's [7] BipedalWalker, LunarLander, CarRacing. (c) Brax [17] Ant and HalfCheetah, Fetch and Humanoid, Grasp and UR5E. (d) Super Mario [3, 50] and RNADesign [48].

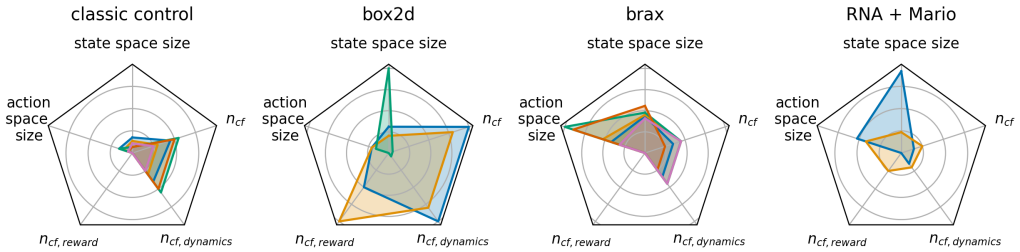


Figure 3: Characteristics of each environment of the environment families showing the action space size, state space size (log-scale), number of context features (n_{cf}), the number of context features directly shaping the reward ($n_{cf,reward}$) and the ones changing the dynamics ($n_{cf,dynamics}$). All axes are scaled to the global extrema and the state space size is additionally on a logarithmic scale.

Action Space We provide both discrete and continuous environments, with six requiring discrete actions and the other ten continuous ones. The actions range from a single dimension to 19.

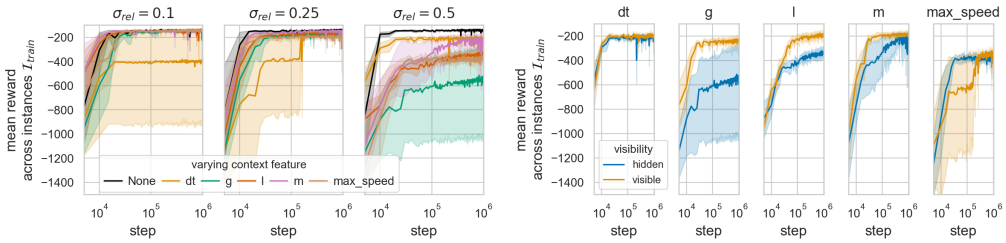
Quality of Reward We cover different kinds of reward signals with our benchmarks, ranging from relatively sparse step penalty style rewards where the agent only receives a reward of -1 each step to complex composite reward functions in e.g. the Brax-based environments. The latter version is also quite informative, providing updates on factors like movement economy and progress towards the goal whereas the former does not let the agents distinguish between transitions without looking at the whole episode. Further examples for sparse rewards are the CARLCartPoleEnv and CARLVehicleRacingEnv.

Context Spaces While the full details of all possible context configurations can be seen in Appendix G, for brevity here we only discuss the differences between context spaces and the configuration possibilities they provide. Depending on the environment the context features have different influence on the dynamics and the reward. Of all 131 registered context features, 98 % influence the dynamics. This means that if a context feature is changed the transition from one state into the other is changed as well. Only 5 % of the context features shape the reward. Most context features (87 %) are continuous, the rest is categorical or discrete. With the explicit availability of context features CARL lends it self to study the robustness of agents by adding noise on top of the specific context features. Further, the provided bounds and sampling distributions of the context spaces that are provided as part of CARL enable better comparability and reproducibility for future research efforts in the realm of general RL agents.

Summary Comparing our benchmarks along these attributes, we see a wide spread in most of them (Figure 3). For the first iteration of CARL, we focused on fairly cheap-to-run problems to lower the barrier of entry as much as possible. Nevertheless, as CARL will further grow over time, the diversity of benchmarks will further increase and we will also include harder benchmarks. Already now, CARL provides a benchmarking collection that tasks agents with generalizing in addition to solving the tasks most common in modern RL while providing a platform for reproducible research.

6 Experiments

Having discussed CARL's theoretical foundation as well as its initial set of benchmarks, we now study several first research questions regarding the effects of context. Our experiments are designed to demonstrate that we can use CARL to gain meaningful insights into the Meta-RL setting even on simple environments. Details about the hyperparameter settings and used hardware for all experiments are listed in Appendix B. In each of them, we train and evaluate on 5 different random seeds and a set of 100 sampled contexts. All experiments can be reproduced using the scripts we provide with the benchmark library at `https://www.github.com/automl/CARL`.



(a) Q1: Different context distributions

(b) Q2: Visible vs. hidden context for $\sigma_{\text{rel}} = 0.5$

Figure 4: Training performance of a DDPG agent on CARLPendulumEnv with (a) different context distributions (Q1) and (b) the effect of visible context (Q2). σ is the standard deviation for sampling the context. The context feature `dt` refers to the observation interval length, `g` to gravity, `l` to the pole length, `m` to the pole mass and `max_speed` to the maximal speed of CARLPendulumEnv.

6.1 Q1: How do Context Features Influence an Agent's Training Performance?

In order to gain an intuition on how context features influence an agent's training performance, we evaluate a DDPG [29] agent on the well known *Pendulum* task from OpenAI gym [7]. Through CARL, we can vary the context features *gravity (g)*, *integration time step (dt)*, pendulum *mass (m)* and *length (l)* as well as the *maximal speed* (see Figure 4). In Equation A.1 in the appendix, we show the dynamic system of Pendulum.

To understand how context features influence an agent's performance, for each considered context feature we sample a set of 100 instances $\{i_k\}_{k=1,\dots,100}$ for each task within the ranges provided by the environment specification while keeping the others context features fixed to their default. Each context feature c of an instance is sampled from a normal distribution, centered around its default value such that $c \sim \mathcal{N}(c_{\text{def}}, \sigma_{\text{rel}} \cdot c_{\text{def}})$ where c_{def} is the default value defined in the original environment and σ_{rel} is the relative standard deviation. Here we evaluate three relative standard deviations $\sigma_{\text{rel}} \in \{0.1, 0.25, 0.5\}$ to show the impact of varying similarities of the instance distribution.

Further, we treat the context as hidden, only implicitly noticeable to the agent through the observation of the state features. While small changes in the context barely have an impact on the training performance of the agent, large variations of a *single context feature* can make the learning task challenging (see Figure 4a). This gives evidence that even simple, cheap-to-run environments can provide an agent with challenging learning tasks, depending on the level of generalization required. Note, this style of training with implicit contexts is currently the default setting for training on vision-based environments such as ProcGen (see Section 4). We refer to a the appendix Section C for a first impression on the influence of context on a vision-based environment, CARLMarioEnv, showing similar insights as for Pendulum.

6.2 Q2: Are Explicit Context Features Necessary to Learn General Agents?

To answer this question we first use the same agent and environment setup, i.e. DDPG with the same hyperparameters on Pendulum and the widest context distribution ($\sigma_{\text{rel}} = 0.5$). Our results (see Figure 4b) suggest that explicitly making agents aware to the change in transition dynamics generally results in a better performance when a generalization over strong deviations in context features is required. This is clearly observable by comparing results for context features that have a higher impact on the final performance, such as *gravity (g)*, *pole length (l)* and *pole mass (m)*. For a fairly low impact context feature *integration time step (dt)*, making the context visible results in a lower standard deviation and a slightly higher final reward. Still, varying *dt* led to minor loss of reward compared to original Pendulum task (black curve in Figure 4a). For the *max_speed* context, both training variants struggle to achieve as high a reward. In the early training stages, the agent trained with access to the context achieved a lower reward than its counterpart. However, in the latter half it could catch up and slightly improve over the context-oblivious agent.

One context feature that heavily influences the dynamics in CARLMarioEnv environment is the inertia of Mario. In Figure 5, we see that a higher variation of the inertia improves the performance of the PPO agent and leads to faster training. This effect can be explained by the influence of Mario's inertia on the exploration of the agent (i.e. a lower inertia makes it easier for the agent to move).

An interesting question for future work is how different context features change the learning behavior of agents and to which degree generalization is impacted by it.

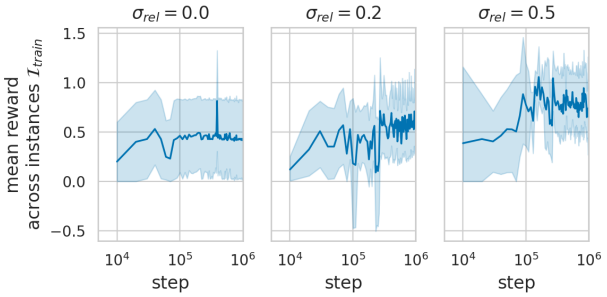


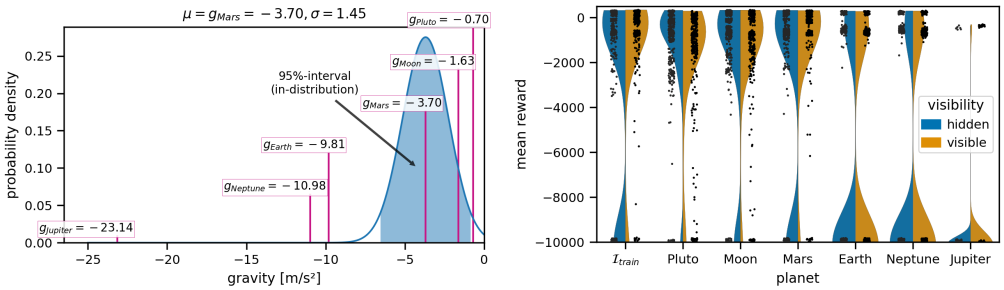
Figure 5: Training performance on CARLMarioEnv where the inertia of Mario is varied .

6.3 Q3: To What Extent Can We Transfer a Learned Policy to a New Context?

To gain insights to what extend the agent is able to transfer a learned policy to a new context we create the "Landing in Space" scenario based on the well known LunarLander environment. To this end, a DQN [35] agent is trained on a rather narrow context distribution. For testing, we then place this agent in a new context which might not have been observed during training.

Landing in Space In this task, the agent is challenged to land a spacecraft on seen as well as unseen planets. We model the different planets by only adjusting the gravities by a well-defined normal distribution and train the LunarLander to land on smaller planets. The train distribution is centered on Mars ($\mu = 3.7\,\mathrm{m/s^2}$) and the standard deviation ($\sigma = 1.45\,\mathrm{m/s^2}$) is chosen such that Mars and Moon are considered as in-distribution whilst Pluto, Earth, Neptune and Jupiter are considered as unseen and out-of-distribution, see Figure 6a. Here we deem planets as *in-distribution* if their gravity is within the $95\,\%$-interval of the training distribution. For training, we sample 100 gravities from this distribution. We use 5 random seeds for training and testing and collect 100 episodes on each planet for both cases where the context is hidden and where the context feature gravity is visible to the agent. Note that although for each test episode on a planet the gravity is fixed and the same, the LunarLander environment generates different initial starting conditions and landscapes to land on. For this reason the lander might still fail to safely land and crash in some cases. To capture crashes and to distinguish them from successful but suboptimal landings, we increase the game over penalty from -100 to -10000 during testing.

In Distribution Generalization As to be expected, the test performances for landing on Mars and Moon are most similar to the Mars-centered training distribution. Agents trained with access to the gravity feature receive higher rewards and less crashes on in-distribution planets than their context-oblivious counterparts, as shown at the bottom of Figure 6b .



(a) Train distribution and test instances (planets)

(b) Reward achieved on different instances

Figure 6: "Landing in Space" scenario: (a) Mars-centered gravity train distribution with in- and out-of-distribution test instances (planets). (b) Resulting rewards for landing on different planets.

Out of Distribution Generalization A more interesting and understudied question in RL is the extend to which agents are capable of generalizing to out-of-distribution tasks. With CARLs possibility to define particular distributions over context features, we can study this question in detail. The agent fails least often on Pluto since (a) it has a gravity still close to the training distribution and (b) has a low gravity. The lower the gravity, the longer the timeframe to land is which creates easier landing scenarios. We can further note that fewer runs on Pluto lead to as high rewards as on in-distribution planets Moon and Mars. This is likely due to agents wasting fuel by anticipating a harder landing, thus burning more fuel to counteract. Interestingly, even for more difficult out-of-distribution planets such as Earth and Neptune we can observe positive landing results for both agents trained with and those without access to the context. However, test performance deteriorates due to more frequent crashing on more high-gravity planets. While we have seen some capability of trained agents to transfer even to out-of-distribution environments, we do not expect vanilla agents to generalize to highly different environments.

7 Further Open Challenges Enabled by CARL

Although these experiments only give a first impression on how CARL can be used to gain novel insights into RL agents, we see many more possibilities for future research involving CARL. We discuss six open challenges and how CARL could be used to tackle these. (I) As CARL provides ground truths for all considered context features it is suitable to study novel agents that separate representation learning from policy learning. (II) CARL will be useful in studying RL agents for uncertain dynamics, by easily perturbing context features. (III) It is particularly suitable for training and evaluation of continual RL methods by continuously adapting context distributions over time. (IV) The ground truth on contexts can also be used to study explainability and interpretability methods of deep RL. (V) With the complexity of modern RL methods, they have become very sensitive to their hyperparameters. CARL' flexibility and focus on generalization enables research into AutoRL methods that optimize agents for generality. (VI) Finally, it is an open question for safe RL whether context information could contribute to decide whether policies are applicable to unseen instances of an environment. Please find a detailed discussion in Appendix D.

8 Limitations and Societal and Ethical Implications

Although in principle some environments of CARL allow to study the impact of context on vision-based agents, our analysis focuses on featurized environments. Thus, we did not study different ways of directly exposing context information to vision-based agents which would require novel architectures to handle this context. We see such experiments and design of novel agents as future work that can follow from using CARL.

Our experimentation limits itself to static contexts and does not consider learning with dynamic contexts or continual learning. We leave this for future studies since generalization to fixed contexts already poses a major challenge. Lastly, we limited our experiments on varying individual context features. Off-the-shelf agents are not yet designed to be adaptive to contexts. Varying individual

features already posed a challenge to learn with for the considered agents. With progress in the field we hope that agents will become more flexible and can handle ever more changes in environments.

We foresee no new direct societal and ethical implications other than the known concerns regarding autonomous agents and RL (e.g., in a military context). However, by trying to lower the barrier of entry for Meta-RL research we hope to i) reduce the required compute for future research, ii) facilitate novel designs of RL agents and iii) reach a more diverse research community.

9 Conclusion

We introduced CARL, a highly flexible benchmark library for enabling studies on generalizable RL via task variations and context features. By employing contextual RL, CARL extends common RL environments by making the context configurable and potentially visible. Besides providing a ready-to-use benchmark library and discussing the role of context in general RL, we ran first experiments to analyse its aspects. Our main insights are that (i) the more the context is varied, the more difficult learning becomes and (ii) making the agent context-aware can facilitate training and increase generalization. In addition, CARL is suitable to study generalization in detail by being able to carefully set instance and context distributions. We provide empirical evidence that current agents can generalize well on in-distribution test instances but fail to do so on out-of-distribution settings. In conclusion, we believe that CARL will be a valuable benchmark to advance on open challenges like generalizing RL, representation and continual learning, safe RL and AutoRL.

Acknowledgements

Carolin Benjamins, Theresa Eimer and Marius Lindauer acknowledge funding by the German Research Foundation under LI 2801/4-1. André Biedenkapp and Frank Hutter acknowledge funding by the Robert Bosch GmbH.

References

[1] Abdolshah, M., Le, H., George, T. K., Gupta, S., Rana, S., and Venkatesh, S. (2021). A new representation of successor features for transfer across dissimilar environments. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1–9. PMLR.

[2] Arel, I., Liu, C., Urbanik, T., and Kohls, A. (2010). Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135.

[3] Awiszus, M., Schubert, F., and Rosenhahn, B. (2020). TOAD-GAN: Coherent style level generation from a single example. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16.

[4] Badia, A., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z., and Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 507–517. PMLR.

[5] Biedenkapp, A., Bozkurt, H. F., Eimer, T., Hutter, F., and Lindauer, M. (2020). Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In Lang, J., Giacomo, G. D., Dilkina, B., and Milano, M., editors, *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*, pages 427–434.

[6] Biedenkapp, A., Marben, J., Lindauer, M., and Hutter, F. (2018). CAVE: Configuration assessment, visualization and evaluation. In *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*, Lecture Notes in Computer Science. Springer.

[7] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*, abs/1606.01540.

[8] Chevalier-Boisvert, M., Willems, L., and Pal, S. (2018). Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`.

[9] Co-Reyes, J. D., Miao, Y., Peng, D., Real, E., Le, Q. V., Levine, S., Lee, H., and Faust, A. (2021). Evolving reinforcement learning algorithms. In *Proceedings of the International Conference on Learning Representations (ICLR'21)*. OpenReview.net. Published online: `iclr.cc`.

[10] Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. (2019). Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*.

[11] Du, S., Krishnamurthy, A., Jiang, N., Agarwal, A., Dudík, M., and Langford, J. (2019). Provably efficient RL with rich observations via latent state decoding. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1665–1674. PMLR.

[12] Duan, Y., Schulman, J., Chen, X., Bartlett, P., Sutskever, I., and Abbeel, P. (2016). Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779.

[13] Eimer, T., Biedenkapp, A., Hutter, F., and Lindauer, M. (2021). Self-paced context evaluation for contextual reinforcement learning. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, volume 139 of *Proceedings of Machine Learning Research*, pages 2948–2958. PMLR.

[14] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. (2018). IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pages 1406–1415. PMLR.

[15] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D. and Teh, Y., editors, *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, volume 70, pages 1126–1135. Proceedings of Machine Learning Research.

[16] Florensa, C., Held, D., Geng, X., and Abbeel, P. (2018). Automatic goal generation for reinforcement learning agents. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1514–1523. PMLR.

[17] Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. (2021). Brax - A differentiable physics engine for large scale rigid body simulation. *CoRR*, abs/2106.13281.

[18] Fu, H., Tang, H., Hao, J., Chen, C., Feng, X., Li, D., and Liu, W. (2021a). Towards effective context for meta-reinforcement learning: an approach based on contrastive learning. In *Proceedings of the Conference on Artificial Intelligence (AAAI'21)*, pages 7457–7465. AAAI Press.

[19] Fu, X., Yang, G., Agrawal, P., and Jaakkola, T. (2021b). Learning task informed abstractions. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3480–3491. PMLR.

[20] Hallak, A., Castro, D. D., and Mannor, S. (2015). Contextual markov decision processes. *arXiv:1502.02259 [stat.ML]*.

[21] J. Parker-Holder, V., Nguyen, S. J., and Roberts (2020). Provably efficient online hyperparameter optimization with population-based bandits. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, volume 33, pages 17200–17211.

[22] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. (2017). Population based training of neural networks. *arXiv:1711.09846 [cs.LG]*.

[23] Kaddour, J., Saemundsson, S., and Deisenroth, M. (2020). Probabilistic active meta-learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20813–20822. Curran Associates, Inc.

[24] Klink, P., D'Eramo, C., Peters, J., and Pajarinen, J. (2020). Self-paced deep reinforcement learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[25] Kostas, J., Chandak, Y., Jordan, S. M., Theocharous, G., and Thomas, P. (2021). High confidence generalization for reinforcement learning. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5764–5773. PMLR.

[26] Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. (2020). The nethack learning environment. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

[27] Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. (2020). Learning quadrupedal locomotion over challenging terrain. *Science in Robotics*, 5.

[28] Li, X., Zhang, J., Bian, J., Tong, Y., and Liu, T. (2019). A cooperative multi-agent reinforcement learning framework for resource balancing in complex logistics network. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS*, pages 980–988. International Foundation for Autonomous Agents and Multiagent Systems.

[29] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

[30] Lu, M., Shahn, Z., Sow, D., Doshi-Velez, F., and Lehman, L. H. (2020). Is deep reinforcement learning ready for practical applications in healthcare? A sensitivity analysis of duel-ddqn for hemodynamic management in sepsis patients. In *AMIA 2020, American Medical Informatics Association Annual Symposium, Virtual Event, USA, November 14-18, 2020*. AMIA.

[31] Machado, M., Bellemare, M., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Intell. Res.*, 61:523–562.

[32] Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. (2020). Teacher-student curriculum learning. *IEEE Trans. Neural Networks Learn. Syst.*, 31(9):3732–3740.

[33] Meng, T. and Khushi, M. (2019). Reinforcement learning in financial markets. *Data*, 4(3):110.

[34] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015a). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[35] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015b). Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533.

[36] Modi, A., Jiang, N., Singh, S. P., and Tewari, A. (2018). Markov decision processes with continuous side information. In *Algorithmic Learning Theory (ALT'18)*, volume 83, pages 597–618.

[37] Morimoto, J. and Doya, K. (2000). Robust reinforcement learning. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 1061–1067. MIT Press.

[38] Nguyen, S., Duminy, N., Manoury, A., Duhaut, D., and Buche, C. (2021). Robots learn increasingly complex tasks with intrinsic motivation and automatic curriculum learning. *Künstliche Intell.*, 35(1):81–90.

[39] Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvári, C., Singh, S., Roy, B. V., Sutton, R. S., Silver, D., and van Hasselt, H. (2020). Behaviour suite for reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[40] Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2817–2826. PMLR.

[41] Raffin, A. (2020). Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`.

[42] Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. (2019). Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`.

[43] Rajan, R., Diaz, J. L. B., Guttikonda, S., Ferreira, F., Biedenkapp, A., and Hutter, F. (2019). MDP Playground: Controlling dimensions of hardness in reinforcement learning. *CoRR*, abs/1909.07750.

[44] Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. (2019). Efficient off-policy meta-reinforcement learning via probabilistic context variables. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, volume 97, pages 5331–5340. PMLR.

[45] Ray, A., Achiam, J., and Amodei, D. (2019). Benchmarking Safe Exploration in Deep Reinforcement Learning.

[46] Rice, J. (1976). The algorithm selection problem. *Advances in Computers*, 15:65–118.

[47] Romac, C., Portelas, R., Hofmann, K., and Oudeyer, P. (2021). Teachmyagent: a benchmark for automatic curriculum learning in deep RL. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, pages 9052–9063. PMLR.

[48] Runge, F., Stoll, D., Falkner, S., and Hutter, F. (2019). Learning to Design RNA. In *Proceedings of the International Conference on Learning Representations (ICLR'19)*. Published online: `iclr.cc`.

[49] Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Kuttler, H., Grefenstette, E., and Rocktäschel, T. (2021). Minihack the planet: A sandbox for open-ended reinforcement learning research.

[50] Schubert, F., Awiszus, M., and Rosenhahn, B. (2021). Toad-gan: a flexible framework for few-shot level generation in token-based games. *IEEE Transactions on Games*, pages 1–1.

[51] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

[52] Seo, Y., Lee, K., Clavera, I., Kurutach, T., Shin, J., and Abbeel, P. (2020). Trajectory-wise multiple choice learning for dynamics generalization in reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12968–12979. Curran Associates, Inc.

[53] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[54] Sodhani, S., Denoyer, L., Kamienny, P., and Delalleau, O. (2021a). Mtenv - environment interface for mulit-task reinforcement learning. Github.

[55] Sodhani, S., Zhang, A., and Pineau, J. (2021b). Multi-task reinforcement learning with context-based representations. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9767–9779. PMLR.

[56] van Rijn, J. and Hutter, F. (2018). Hyperparameter importance across datasets. In Guo, Y. and F.Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2367–2376. ACM Press.

[57] Wang, J., King, M., Porcel, N., Kurth-Nelson, Z., Zhu, T., Deck, C., Choy, P., Cassin, M., Reynolds, M., Song, H., Buttimore, G., Reichert, D., Rabinowitz, N., Matthey, L., Hassabis, D., Lerchner, A., and Botvinick, M. (2021). Alchemy: A structured task distribution for meta-reinforcement learning. *CoRR*, abs/2102.02926.

[58] Wang, J., Kurth-Nelson, Z., Soyer, H., Leibo, J., Tirumala, D., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2017). Learning to reinforcement learn. In Gunzelmann, G., Howes, A., Tenbrink, T., and Davelaar, E., editors, *Proceedings of the 39th Annual Meeting of the Cognitive Science Society*. cognitivesciencesociety.org.

[59] Xu, L., Hoos, H., and Leyton-Brown, K. (2010). Hydra: Automatically configuring algorithms for portfolio-based selection. In Fox, M. and Poole, D., editors, *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'10)*, pages 210–216. AAAI Press.

[60] Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. (2021). Reinforcement learning with prototypical representations. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11920–11931. PMLR.

[61] Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2019). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*.

[62] Zhang, A., Sodhani, S., Khetarpal, K., and Pineau, J. (2021a). Learning robust state abstractions for hidden-parameter block mdps. In *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net.

[63] Zhang, H., Chen, H., Boning, D., and Hsieh, C. (2021b). Robust reinforcement learning on state observations with learned optimal adversary. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

[64] Zhou, W., Pinto, L., and Gupta, A. (2019). Environment probing interaction policies. In *Proceedings of the International Conference on Learning Representations (ICLR'19)*. Published online: `iclr.cc`.

[65] Zhou, Z., Li, X., and Zare, R. (2017). Optimizing chemical reactions with deep reinforcement learning. *ACS central science*, 3(12):1337–1344.

Part VI

Conclusion

Chapter 14

Summary and Discussion

In part I we provided a general motivation for the dissertation and defined the research goals. To answer the question *"Can we learn general dynamic configuration policies using reinforcement learning?"* we identified four key challenges each with its own research questions. We addressed these challenges and questions in the four core parts of this dissertation:

Part II: Dynamic Algorithm Configuration: The Problem

Part III: Dynamic Algorithm Configuration: Case Studies

Part IV: Dynamic Algorithm Configuration: Benchmarking

Part V: Improving RL From the Lens of DAC

We address the first key challenge in Part II by providing the first formal problem definition of dynamic algorithm configuration. Of particular importance for this dissertation is Chapter 5. This chapter builds the cornerstone of this dissertation. Besides the formal problem definition, this chapter proposes a way of modeling dynamic algorithm configuration as a contextual Markov decision process and identifies reinforcement learning as the go-to solution approach. This way of modeling the problem is used throughout the dissertation and all solution approaches we explored build on reinforcement learning. Further, we provide the first benchmarks for studying dynamic configuration in a principled manner and provide the first steps towards a standardized interface for dynamic algorithm configuration (DAC) benchmarks. It is worth noting that the published version of this chapter has had a high impact on the community and has already resulted in and inspired multiple follow-up publications. Chapter 6 extends the preceding chapter by discussing further methods for DAC besides reinforcement learning. Further, Chapter 6 provides a discussion of current limitations of dynamic algorithm configuration and the so far explored solution approaches.

In Part III we go beyond the largely theoretical considerations of the previous part and evaluate dynamic algorithm configuration in representative case studies. This part of the dissertation is largely concerned with addressing the second key challenge of using DAC for real applications. Thus, with this part, we not only tackle research questions but also engineering ones. Chapter 7 introduces dynamic algorithm configuration for step-size adaptation in an evolutionary algorithm. We define and discuss all parts that are necessary to model this as a DAC problem and discuss how we can use RL to learn configuration policies. Already in this early work, we propose how to improve the considered RL method to better suit our needs for DAC. The work provides a large-scale empirical analysis and we show that the learned policies are capable of outperforming a commonly used, hand-crafted baseline policy. Further, we show the generalization capabilities of our learned policies

that are possible through the use of context information. Chapter 8 provides a second in-depth case-study. In this study, we discuss how to use DAC for AI planning. Similar to the preceding chapter we define and discuss all relevant components that are necessary to be able to model this as a DAC problem and use RL to solve it. An important distinction here is that additional consideration needed to be made to enable interfacing of two algorithms that are written in different languages. Having defined all components we prove that DAC subsumes prior meta-algorithmic frameworks. Our large empirical evaluation confirms the theoretical insights and demonstrates that DAC is a powerful tool in this setting by outperforming even the theoretical best (static) algorithm selector, something no prior framework was capable of achieving.

Part IV focuses on addressing the third key challenge in the form of reproducibility. While the previous parts have demonstrated that we can learn very capable and general configuration policies, reproducibility is a key factor in ensuring meaningful progress in the field. Chapter 9 builds on the previous chapters and consolidates all benchmarks we introduced previously into a large collection of benchmarks which we dub DACBench. Further, it provides the first standardized interface of these benchmarks to lower the barrier of entry for new researchers. Besides the benchmarks from our previous works, we identify interesting benchmarks from prior works. All in all, DACBench provides a diverse set of DAC benchmarks starting with cheap-to-run toy benchmarks up to more complex problem settings, such as configuration of multiple parameters of an evolutionary algorithm. However, in its first version, DACBench did not provide any ground truth on the considered benchmarks. Chapter 10 rectifies this by introducing novel benchmarks into DACBench for which it is possible to provide ground truth performances and thus compute the true optimal policies. This benchmark thus is ideal to study the behavior of potential DAC solution approaches. We show this by exemplary evaluating the current limitations of a DDQN, our most used solution method. This study shows that, while DDQN is capable of learning the optimal policies, with the considered hyperparameter settings, DDQN starts to struggle with larger problem sizes and potential parameter configuration values.

The last core part of the dissertation, Part V considers RL from the lens of DAC to address the last key challenge of generalization in RL. Chapter 11, inspired by insights from previous chapters, proposes to dynamically configure temporal exploration of deep reinforcement learning agents. Such agents are capable of learning temporal connections and thereby *when* it is necessary to make new decisions. This is key to making DAC solution approaches more sample efficient as often actions need to be repeated for multiple time steps. We demonstrate that TempoRL agents are capable of learning quicker and sometimes even reach better final performance than standard RL counterparts. Chapter 12 provides the first approach to improve RL algorithms when learning across multiple environments. To this end, we propose to use a self-paced learning approach where the agent can decide over time which problem instances it wants to train on. Our evaluation shows that this approach allows not only faster training but also results in better generalization of the learned policies. As DAC by design considers multiple problem instances, this work could facilitate more sample-efficient DAC solution approaches. Finally, in Chapter 13 we provide an in-depth discussion of contextual reinforcement learning. Throughout the dissertation, we have used context information to enable RL agents to train across multiple environments and to be able to generalize to unseen environments. Still, in our work so far we have used standard reinforcement learning methods. This is mostly due to RL research having, so far, not focused on learning general agents but rather focused on single environment settings. To facilitate research on generalization in RL this last chapter proposes a novel benchmark specifically designed for contextual reinforcement learning. While this benchmark is related to DACBench, it provides more fine-grained control over the problem instances. In DACBench, the differences across problem instances are not easily controllable and the available context features often only provide imperfect information about the problem

instances. In our novel benchmark CARL, context features directly characterize how the problem instances differ from each other and provide perfect information. We believe that any advancements that will be made on the CARL benchmark will feed back into DAC research by providing novel ways of considering context features in RL and by extension DAC.

Chapter 15

Lessons Learned for a new Research Field

Our work on dynamic configuration paved the way for a new research field. Along the way, we learned many important lessons that often arose due to new problems that we did not expect from the get-go. Here we will summarize these important lessons to inform future research on DAC but also on other fields in AI.

Stepping Out of the Comfort Zone When setting out on tackling new problems it is important to not only look at the happenings in one's own community. Often, different communities face the same or related challenges. In our specific case, as discussed in Chapters 3 and 6, dynamic configuration problems arise in many communities. All these communities proposed some forms of tackling these problems. While most of these works focus on manual efforts on dynamic configuration they provide vital information which can be used for automating such processes.

In the simplest case, prior works can inform us which parameters need to be treated dynamically and concrete examples provide information that can help in setting up learning environments for DAC policies. As a consequence, DACBench does not only contain benchmarks of our creation but also ones that have been considered by prior work. Besides increasing the diversity of available benchmarks in DACBench, this provides us with interesting benchmarks for which we did not control all design decisions. This also allowed us to confirm if our proposed standardized interface was compatible with research that was not of our creation.

All in all, stepping out of one's comfort zone and looking beyond the edge of one's community is highly beneficial. It might uncover problems and challenges that might have not yet had to be faced by one's own community. Thus, taking these challenges into account allows us to essentially plan ahead.

The Importance of Engineering for Reproducibility and Benchmarking It comes as no surprise that standardized benchmarks are beneficial to the research community. However, when starting our research on DAC there were no standardized benchmarks or interfaces to consider for evaluation.

In the beginning, our progress was slowed down by having to face new engineering challenges. For example, it is easy to propose to dynamically configure some parameter in AI planning. Setting out to do so however provided the challenge of interfacing two systems that are designed in different programming languages. This problem was even worsened by the fact that most algorithms are designed to run uninterrupted. So as a first step it was important to identify a "point of entry" for our interface. Partly as an engineering challenge, partly as a research question, we needed to address which information needs to be exchanged via this entry point and how to send around this information. The first few tries required weeks of setting up such an interface and making sure that it is robust

enough without throwing any unexpected errors. The final interface proved to be easy to understand and is workable with new algorithms. For example, in a, as of yet, unpublished work setting up the interface to an algorithm in a language we did not interface with prior only required an afternoon of work instead of multiple weeks.

By now, having designed toy benchmarks and interfaces for the considered case studies, the interface proves to be robust and easy to use for a variety of algorithms. Further, all this work is consolidated in the first standardized collection of benchmarks. This enables us to focus future research efforts on evaluating novel solution approaches, rather than having to spend much effort on designing new interfaces. Further, DACBench also makes it very easy to reproduce research results. This is especially important for solution approaches based on reinforcement learning. RL research often does not facilitate easy reproducibility (Henderson et al., 2018) which hinders progress in the field. As RL is a crucial component in our research on DAC, it is thus even more important to ensure the reproducibility of our work. Overall, this also lowers the barrier of entry for new researchers interested in DAC problems and solution approaches.

The Benefit of Working on Real World Applications DAC provides an important playground for future RL research as it confronts RL researchers with real-world problems. Current standard problem settings in RL only consider learning policies for single problem instances which have straightforward properties. Typically, RL solution approaches are nearly always evaluated on (video) game-playing problem instances. While RL has shown incredible successes on such problems, research has often progressed without taking properties into account that one might face in the real world. This is most obviously demonstrated as learned game-playing policies are nearly always evaluated in the same setting as they were trained in. Such policies thus can not be expected to generalize well even if they might produce good results on a single problem instance. However, the reported successes of RL inspire many potential users to try out RL for their problem domain. Such users are often left with the impression that RL "does not work" as the problem setting they want to consider barely resembles the typical problem considered in RL. Counter to this often repeated claim, throughout this dissertation we have demonstrated that deep RL methods are capable of learning general policies on real-world problems with comparatively little required compute (if compared to the millions of required training steps when learning to play video games). Still, this did not work out of the box and we first needed to build expert knowledge to set up the learning agents properly.

Further, our work on DAC has enabled us to discover shortcomings of RL methods and how to address them (Part V is wholly dedicated to this cause). We proposed a method to move from a fairly reactive way of doing reinforcement learning to a more proactive way and further proposed a way of self-paced learning for contextual RL. All in all, with the focus on generalization for DAC, we paved the way for research on contextual RL and generalization in RL. In the future, we believe DAC could thus be an interesting benchmark and application area for AutoRL systems and novel algorithms with a focus on generalization.

Chapter 16

Future Work

With this dissertation, we have laid the foundation for dynamic algorithm configuration, in particular by means of reinforcement learning. We consider this the beginning of a novel research field that provides ample new research directions to embark upon. Following the work we presented in the preceding parts, we identified the following areas as especially promising research directions for potential future work.

Warmstarting and Learning From Handcrafted Policies The DQN methods we considered in the preceding chapters always needed to learn a value function and policy from scratch. This often creates a significant overhead whereas a warmstarted DQN or one that can learn from existing policies could explore meaningful policies much quicker. Most approaches that we are aware of that consider incorporating prior knowledge into RL agents require some form of imitation learning or behavior cloning.

One of our earliest works on DAC (see Chapter 7) has considered how to warmstart the search for well-performing policies. To this end, we employed the guided policy search method (GPS). While this approach was capable of learning well-performing policies and provided good anytime performance by first learning to imitate a hand-crafted policy, GPS is not a method that can be used out of the box. GPS originates from the robotics community where policy search has to adhere to safety constraints to avoid breaking a robot's limbs during learning. Thus, it has design elements that make it cumbersome to use and could potentially lead to numerical instabilities while training.

Further, vanilla GPS only allowed for an initial imitation learning phase from a single teacher and afterward completely disregards the teacher policy. We rectified that with our extension to GPS that continuously queries the hand-crafted policy. Still, this version only allows to warmstart and continuously learn from a single teacher policy. This requires prior knowledge about which single policy to use as a potential teacher. In several domains there exist many potential policies that could serve this purpose and come with complementary strengths and weaknesses. Thus, to make the best use of GPS (and of our extension), one would first need to do some form of algorithm selection to decide which policy to use for the imitation learning phase(s). This is a problem for all methods that rely on some form of imitation learning or behavior cloning.

Thus, future work in this direction would ideally be agnostic to the reinforcement learning method (unlike warmstarting in GPS), able to learn from multiple teachers and automatically determine which teachers to learn from out of a potential faculty.

Better Methods for Contextual Reinforcement Learning Throughout this dissertation, we have demonstrated that using context information allows us to learn RL agents that are capable of generalizing to different environments. Still, approaches to cRL so far have been rather limited and nearly always only considered a few context features which provide

perfect information about the problem instance (i.e., environment) at hand. For example in our work on SPACE (see Chapter 12) we considered the pole length of a CartPole environment or a flattened representation of a maze. Both types of context let us perfectly distinguish between problem instances and we only needed a small set of context features to do so. As a consequence, in most considered settings it is a perfectly valid approach to concatenate context features to state features when learning policies. However, in more practical settings, such as DAC, we often do not have perfect information available and we might require many more noisy features to be able to adequately distinguish between instances. We strongly believe that, in such cases, simply concatenating context and state features complicates the learning problem.

To mitigate this problem, another line of future work should concern itself with designing novel cRL algorithms that have dedicated ways of dealing with context features. We have already begun the first steps in that direction. In the pure contextual RL setting, we recently evaluated a method for learning a gating mechanism that modulates a selected action based on the available context and made use of our CARL benchmark (see Chapter 13) to study the influence of context on the learning setting (Benjamins et al., 2022). In the more noisy DAC setting, following up on our work on DAC for AI planning (see Chapter 8), we proposed ways of incorporating 305 imperfect context features that describe possible AI planning problem instances (Biedenkapp et al., 2022b). Our results indicate that decoupling learning with context features from learning with state features results in more robust policies. Still, more work in this direction is needed to design dedicated cRL algorithms that are capable of learning from any type of context feature.

DAC for AutoRL While we have demonstrated that modern RL methods, in particular DQN variants, are capable of learning meaningful DAC policies, training them is far from trivial. Due to the brittleness of RL algorithms (Henderson et al., 2018), whenever some failure mode occurs, there are multiple potential causes. Debugging these failures is far from trivial and often even more complicated due to unforeseen interaction effects of some hyperparameters. With expert domain knowledge and extensive manual labor, it is possible to fix such issues, however relying only on experts to manually fix such issues severely limits DAC research if it is to be done via RL. This manual labor could be offloaded to AutoRL (Parker-Holder et al., 2022) methods. However, current approaches to AutoRL require far more compute resources than many research labs have at their disposal.

To mitigate this problem, DAC methods could be designed that can learn to configure the RL algorithm during training. In Chapter 11 we have presented work along these lines. In essence, TempoRL learns how to configure the exploration component over a longer time frame while the underlying RL algorithm is learning the normal behavior policy. Still, in our work, we have not evaluated if TempoRL policies are transferable to novel environments or even algorithms from the same family (such as transfer from DDQN to DQN). While TempoRL presents an important first step in using DAC to configure RL algorithms on the fly, more work has to be done to make DAC an integral component of RL algorithms. It has been demonstrated that dynamic adaptation of the learning rate and discounting factor result in simpler learning problems such that the overall learning speed is increased (François-Lavet et al., 2015). Thus, we believe that DAC is perfectly suited for learning policies for these parameters.

Another possibility with DAC for AutoRL is to learn general configuration policies on a large set of benchmarks. Such pre-trained policies could then be transferred to novel training settings and adapted to the context at hand. Further, similar to current trends in large language models (see, e.g., Brown et al., 2020), pre-trained policies might be suitable for fine-tuning on novel tasks that they have not been exposed to before. This pre-training and fine-tuning paradigm has the advantage that fine-tuning is often computationally

cheaper and thus feasible for smaller research institutions.

All in all, work on AutoRL will feed back into DAC and enable many more users to solve DAC problems by RL without the need for extensive expert knowledge. If AutoRL is achieved by DAC itself, the benefit might be even larger as insights from DAC for AutoRL might inform more research on DAC in general as well as result in more usable RL methods for DAC.

A Solver Designed for DAC Previous meta-algorithmic research has found widespread success, in part to its usability. Tools such as SMAC (Hutter et al., 2011; Lindauer et al., 2022), GGA (Ansótegui et al., 2009) or iRace (López-Ibáñez et al., 2011) and similar AutoML tools, such as auto-sklearn (Feurer et al., 2019) make it relatively easy for non-experts to use the tool for their desired target application. The use of current DAC methods is still fairly involved. In part, this is due to the brittleness of RL, as explained in the previous paragraph, but also due to the non-existence of ready-to-use libraries of cRL algorithms. In all our research we have mostly relied on our own implementations of RL algorithms such that we could easily modify them to work as needed. While we have open-sourced all of our code, we have yet to build a single DAC solver based on all insights from our previous research. The existence of such a tool would make it easier to evaluate novel methods and to better identify strengths and weaknesses of existing methods. This line of work would go hand-in-hand with our work on DACBench (see Chapter 9) which provided a standardized interface for DAC problems and a purpose-built DAC solver could be used to provide novel baselines on DACBench.

With all these potential future works we believe there are exciting works to come in the near and far future of DAC. We are excited to see how the field will mature and to continue contributing and pushing the horizons of DAC.

Appendices

Appendix A

Appendix for Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework

Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework — Supplementary —

André Biedenkapp[1] and **H. Furkan Bozkurt**[1] and **Theresa Eimer**[3] and **Frank Hutter**[1,2] and **Marius Lindauer**[3]

A Appendix

A.1 Importance of Temporal Information for Luby

For a sequence like *Luby*, an agent can benefit from additional information about the sequence, such as the length of the sequence. For example, imagine an agent has to learn the Luby sequence for length $T = 16$. Before time-step 8 the action value 3 would never have to be be played. For a real algorithm to be controlled, such a temporal feature could be encoded by the iteration number directly or some other measure of progress. The state an agent can observe therefore consists of such a time feature and a small history over the five last selected actions.

A.2 Luby Instance Sampling Strategies

Depending on the sampling strategy, the instances can be chosen to be more homogeneous or more heterogeneous. To generate homogeneous instances, we sample a small temporal error $i \sim \mathcal{N}(0, 0.15)_{-0.5 \leq i \leq .5}$ from a two-sided truncated normal distribution. This temporal error is added every step to t. After m steps $t + |\sum_{j=0}^{m} i| > t$ and therefore an element in the Luby sequence will be skipped ($i > 0$) or repeated ($i < 0$). Formally

$$\text{luby}(t, i) = \begin{cases} l_t & \text{if } mod(t, m) \neq 0 \\ l_{t-1} & \text{if } i < 0 \\ l_{t+1} & \text{otherwise} \end{cases} \tag{6}$$

where l_t is the t-th value of the Luby sequence (Equation 7 in the main paper). The resulting instances will overlap at most time-steps, giving a very homogeneous distribution of instances.

To generate heterogeneous instances, we sample different starting points $i \sim \mathcal{N}(L, 0.25)_{0 \leq i \leq T-L}$ of the Luby sequence from a two-sided truncated normal distribution. Every instance is therefore a subsequence of the original Luby sequence. Formally

$$\text{luby}(t, i) = l_{t+i} \tag{7}$$

where l_t is the t-th value of the Luby sequence (Equation 7 in the main paper) These different starting points cause most instances to have little overlap with other sequences.

[1] University of Freiburg, Germany,
email: {biedenka, bozkurf, fh}@cs.uni-freiburg.de
[2] Bosch Center for Artificial Intelligence, Germany
[3] University of Hannover, Germany, email: lastname@tnt.uni-hannover.de

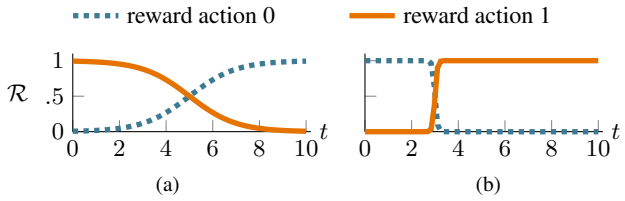


Figure 1: Example rewards for Benchmark *Sigmoid* with $T = 10$, $N = 1$ and $a_{0,t} \in \{0, 1\}$ on two instances, where $p_{(a)} = 5$, $s_{(a)} = 1$ on (a) and $p_{(b)} = 3$, $s_{(b)} = -20$ on (b). The solid/dashed line depicts the reward for action 1/0 at time-step t. On (a) it is preferable to select action 1 for the first halve of the sequence whereas on (b) it is better to start with action 0.

A.3 Sigmoid Example Policies

Figure 1 depicts how different scaling factors and inflection points can be used to construct different problem instances that require different optimal policies. In Figure 1a, the optimal policy is to play action 1 for the first 4 to 5 time-steps before switching over to action 0. In Figure 1b, the scaling factor is inverted, requiring to first play action 0 and then action 1. As the inflection point is shifted to the left the optimal policy selects this action for first 2 to 3 time-steps before playing action 1 the remaining steps.

A.4 Context-Oblivious Agents

Adriaensen and Nowé (2016) proposed two additional context-oblivious agents, (i) PURS, which selects a previously not selected action uniformly at random; otherwise, actions are selected in proportion to the expected number of remaining steps; (ii) GR, which selects an action greedily based on the expected future reward. In the following we discuss why we did not consider PURS and GR.

PURS PURS leverages information about the expected trajectory length, but it does not include the observed reward signal in the decision making process. For tasks where every execution path has the same length (e.g. $Luby_{L=T}$ and *Sigmoid*), PURS would fail to produce a policy other than a uniform random one. Further, when using PURS, we need to have some prior knowledge if shorter or longer trajectories should be preferred. For example on benchmarks like $Luby_{L<T}$, PURS is only able to find a meaningful policy if we know that longer sequences produce better rewards. Thus, we do not consider PURS further.

GR GR is comparable to an ϵ-greedy agent with constant $\epsilon = 0$. This makes GR very prone to getting stuck in local optima which can happen in our experiments quite often.

A.5 Details on Experimental Setup

DQN Details We used a double DQN in chainerRL (0.7.0), i.e., where the target network is updated every 10 episodes and the exploration fraction ϵ of the DQN is linearly decreased from 1.0 to 0.1. We used a fully connected Q-function with a training batch size of 32. In each training iteration only one episode is observed. We chose a network size of 1 layer with 50 hidden units as we only deal with very small state-vectors with at most 8 state features.

Hardware All experiments were run on a compute cluster with nodes equipped with two Intel Xeon E5-2630v4 and 128GB memory running CentOS 7.

A.6 Further Experimental Results

Effect of Short Effective Sequence Length Further experiments for the effect of short effective sequences are given in Tables 1 and 2b, with corresponding reward curves given in Figures 3 and 4 respectively. Table 1 corresponds to Table 1 in the main paper and additionally present the standard error for all presented agents. Table 2b shows similar results with a much higher noise level. In this setting however ϵ-greedy does not achieve the best AUC for short-effective sequences with $L = 32$. From Figure 4 we can observe that, although PS-SMAC results in a better AUC, it alreay converged to a suboptimal sequence, whereas the ϵ-greedy agent is still improving its reward and resulting in a slightly higher final reward.

Stochasticity of Reward Signal Further results for the effect of the stochasticity of the environment are given in Table 3a and 3b with the corresponding plots in Figures 5a and 5b respectively. Table 3 corresponds to Table 2 in the main paper and additionally gives the standard errors of all discussed methods. Figures 5a and 5b show that, no matter the noise level, the ϵ-greedy agent always outperforms both PS-SMAC and URS not only in final reward but also in anytime reward.

Effect of Self-Paced Learning In Figure 2 the trainings performance of SPL on only the selected instances is given. The curve corresponds to Equation (5) of the main paper. The agent starts training on an instance which gives it better initial performance. The agent continues its training on instances which resemble the first initial instance which gave it good performance. After around 5×10^3 training episodes, the agent also includes instances which require opposite policies into its training procedure. It quickly recovers from this shift in instances before it approaches the optimal possible reward.

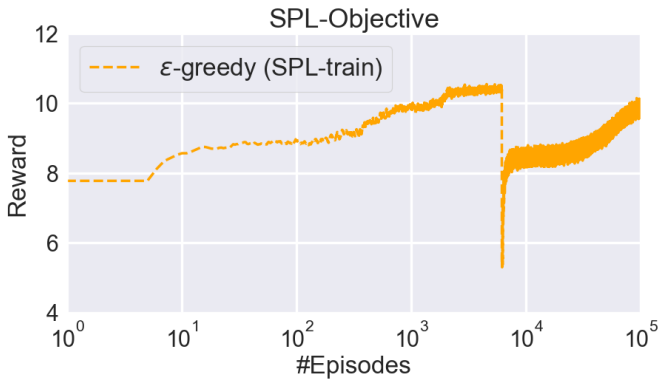


Figure 2: Training performance on instances selected by SPL for 1D-Sigmoid with binary actions and $T = 11$.

	8	16	32
ϵ-greedy	0.86 ± 0.04	0.72 ± 0.05	0.47 ± 0.04
PS-SMAC	0.62 ± 0.10	0.39 ± 0.01	0.39 ± 0.01
URS	0.17 ± 0.02	0.17 ± 0.02	0.17 ± 0.03

(a) Homogeneous

8	16	32	
ϵ-greedy	0.89 ± 0.03	0.75 ± 0.05	0.47 ± 0.03
PS-SMAC	0.56 ± 0.09	0.37 ± 0.01	0.37 ± 0.01
URS	0.17 ± 0.02	0.17 ± 0.02	0.17 ± 0.02

(b) Heterogeneous

Table 1: Results for the discussed agents on *Luby* with fuzzy rewards for $L \in \{8, 16, 32\}$ with $T = 64$ on two instance distributions and a noise factor such that roughly 15% of the actions returned a false positive reward. Results for homogeneous instances are shown in (a) and for heterogeneous instances in (b). The values represent the normalized area under the learning curve for 10^5 training episodes.

	8	16	32
ϵ-greedy	0.76 ± 0.06	0.56 ± 0.10	0.35 ± 0.05
PS-SMAC	0.66 ± 0.10	0.38 ± 0.01	0.38 ± 0.01
URS	0.17 ± 0.04	0.17 ± 0.04	0.17 ± 0.03

(a) Homogeneous

8	16	32	
ϵ-greedy	0.80 ± 0.05	0.59 ± 0.07	0.35 ± 0.05
PS-SMAC	0.55 ± 0.10	0.37 ± 0.01	0.38 ± 0.01
URS	0.16 ± 0.03	0.16 ± 0.05	0.16 ± 0.04

(b) Heterogeneous

Table 2: Results for the discussed agents on *Luby* with fuzzy rewards for $L \in \{8, 16, 32\}$ with $T = 64$ on two instance distributions and a noise factor such that roughly 25% of the actions returned a false positive reward. Results for homogeneous instances are shown in (a) and for heterogeneous instances in (b). The values represent the normalized area under the learning curve for 10^5 training episodes. The corresponding plots are contained in the supplementary material.

	$p(r_t > 0)$				
	0.01	0.08	0.15	0.20	0.25
ϵ-greedy	0.96 ± 0.01	0.92 ± 0.03	0.86 ± 0.04	0.81 ± 0.04	0.76 ± 0.05
PS-SMAC	0.71 ± 0.09	0.63 ± 0.10	0.62 ± 0.10	0.62 ± 0.11	0.55 ± 0.10
URS	0.21 ± 0.01	0.18 ± 0.02	0.17 ± 0.02	0.17 ± 0.02	0.16 ± 0.02

(a) Homogeneous

ϵ-greedy	0.97 ± 0.01	0.94 ± 0.02	0.89 ± 0.03	0.84 ± 0.04	0.80 ± 0.05
PS-SMAC	0.60 ± 0.10	0.63 ± 0.11	0.56 ± 0.09	0.61 ± 0.10	0.52 ± 0.10
URS	0.21 ± 0.02	0.19 ± 0.02	0.17 ± 0.02	0.17 ± 0.03	0.16 ± 0.03

(b) Heterogeneous

Table 3: Sensitivity analysis of the presented agents for varying degrees of noise on Luby. The short effective sequence was set to 8 with a maximal length of 64. The values represent the normalized area under the learning curve for 10^5 training episodes. The corresponding plots are contained in the supplementary material.

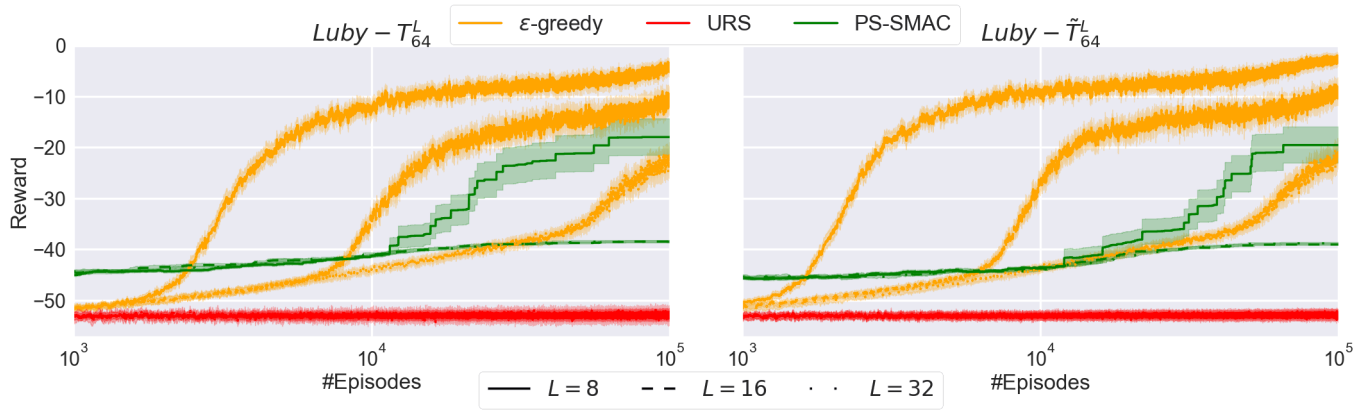


Figure 3: Comparison of ϵ-greedy, URS and PS-SMAC on Luby for varying short effective sequences. (Left) Results on the homogeneous version of Luby. (Right) Results on the heterogeneous version of Luby. Gaussian noise was added to the reward such that roughly 15% of the reward signals gave a false positive signal.

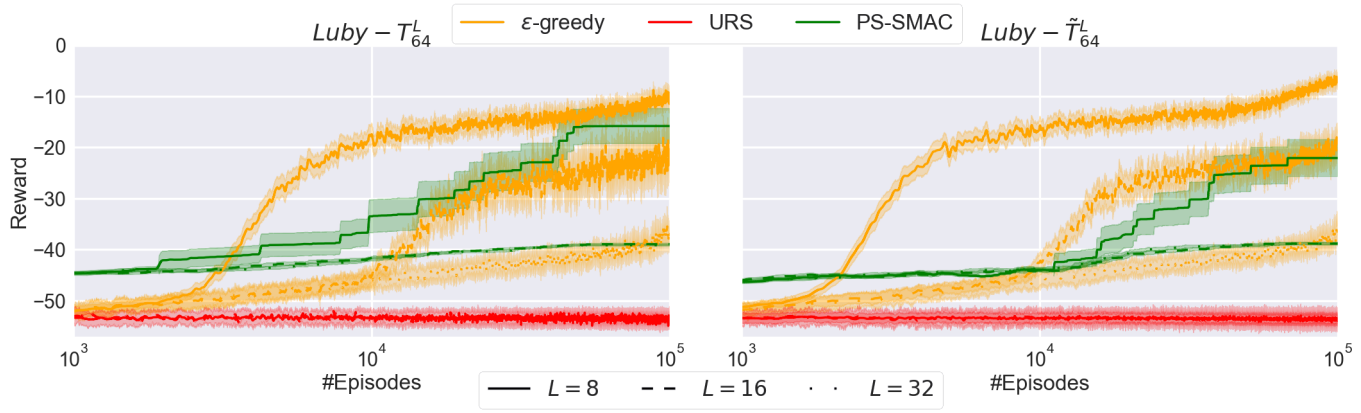
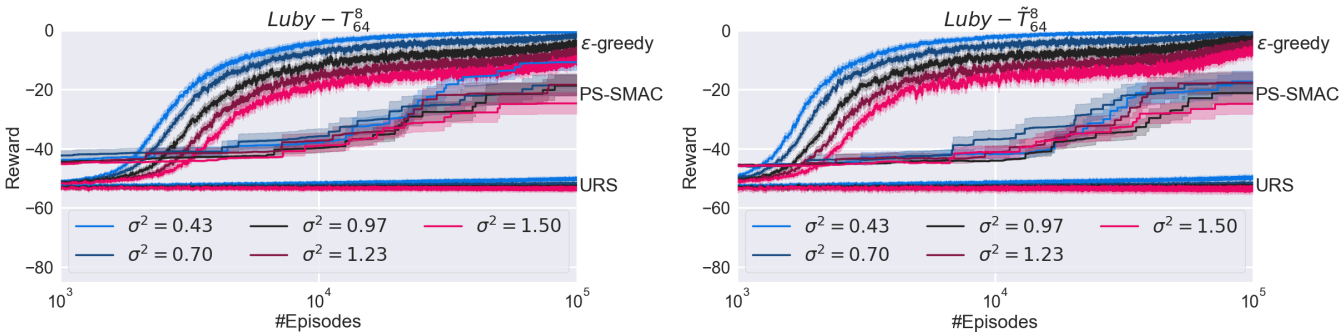


Figure 4: Comparison of ϵ-greedy, URS and PS-SMAC on Luby for varying short effective sequences. (Left) Results on the homogeneous version of Luby. (Right) Results on the heterogeneous version of Luby. Gaussian noise was added to the reward such that roughly 25% of the reward signals gave a false positive signal.



(a) (b)

Figure 5: Sensitivity analysis on Luby (left homogeneous, right heterogeneous) for varying degrees of noise levels.

Appendix B

Appendix for Automated Dynamic Algorithm Configuration

Appendix A. Problem-Theoretical Perspective on DAC

In this Appendix, we present a theoretical motivation of why DAC (Definition 3) is a problem worth studying. In doing so, we provide grounding for many of the higher-level discussions in the main text. Since this kind of analysis is hardly standard, we start by introducing some fundamental concepts (Section A.1), then discuss the main results (Section A.2), and end with the formal justification (Section A.3).

A.1 Fundamental Concepts

A.1.1 Computational Problems

In this work, we formalized dynamic algorithm configuration (DAC) and related computational problems as follows:

> **Definition 4: Computational Problem**
>
> In a computational problem $(\mathcal{X}, \mathcal{R})$, given any input $x \in X$, we are to compute an output y satisfying $(x, y) \in \mathcal{R}$.

Conceptually, each $(x, y) \in \mathcal{R}$ represents a problem instance x and a solution y thereof. Note that instances may have more than one admissible solution, or even none at all. We will also use $R(x) = \{y|(x, y) \in R\}$ to denote the solution set for x. All problem definitions in this paper are structured syntactically as "Given x find a $y \in R(x)$".

Digression on Problem Classes: Problems of this form are also known as "search problems". To avoid confusion, problem *classes* group problems (e.g., DAC), not problem instances (e.g., DAC scenarios), i.e., when viewing DAC as a search problem (as in Definition 3), $\mathcal{X}$ would correspond to the set of all possible DAC scenarios and $\mathcal{R}(x)$ the set of optimal policies for some DAC scenario x. The choice to restrict ourselves to search problems was a trade-off between (i) theoretical convenience / simplicity and (ii) expressiveness, i.e., alternative formulations exist that better model many of the problems we consider:

Optimization: Can express that not every solution is equally good. For instance, in this work we use (i) "find x satisfying $x \in \arg\max_x f(x)$" rather than (ii) "find x maximizing $f(x)$". Note that this difference, while subtle, is important for problem theory: For instance, if $\arg\max_x f(x) = \emptyset$, in (i) we should return that no solution exists, while in (ii) we should return an as good as possible solution.

Distributional: Can express that all inputs are *not* equally likely, by modelling inputs as a distribution $\mathcal{D}$ rather than a set $\mathcal{X}$. Note that while we do not use distributional problems on the meta-level, we do use them as target problems.

However, for these problem classes, standard definitions for theoretical concepts such as *reducibility* do not exist and any satisfactory definition would significantly complicate the reduction proofs in this appendix.

A.1.2 Reducibility

Problem formalization enables formal reasoning about the relationship between problems. In this appendix, we focus on a specific kind of relationship: Reducibility, as defined by Papadimitriou (1994, p. 506):

Definition 5: Many-one Reducibility (m-reducibility)

Let (X, R) and (X', R') be two computational problems, with $R : X \times Y$ and $R' : X' \times Y'$. We say that (X, R) is many-one reducible to (X', R'), or also m-reducible, which we denote $(X, R) \leq_m (X', R')$, if and only if computable functions

`formulate:` $X \to X'$

`interpret:` $X \times Y' \to Y$

exist such that $\forall\ x \in X$ holds:

1. $(\texttt{formulate}(x), y') \in R' \implies (x, \texttt{interpret}(x, y')) \in R$
2. $R'(\texttt{formulate}(x)) = \emptyset \implies R(x) = \emptyset$

Conceptually, (1) all solutions of the reformulated problem instance can be interpreted as a solution to the original problem instance, and (2) if the reformulated problem instance does not have any solutions, the original problem instance should neither.

When a problem R is m-reducible to another R', R can be *solved by reduction* to R', i.e., given an algorithm a' for R', $a(x) = \texttt{interpret}(x, a'(\texttt{formulate}(x)))$ is an algorithm for R. It is worth noting that the existence of a many-one reduction does not necessarily render R irrelevant: Solving R by reduction to R' may inherently increase computational *complexity* since (i) the reduction itself (i.e., `formulate`, `interpret`) may be costly (ii) the reduction may abstract relevant info, making the reduced problem harder to solve. The practical relevance of a reduction is further limited by the performance of known algorithms for R'.

Note that sometimes, even though one problem is not generally reducible to another, a special case is. We define this notion as

Definition 6: Conditional Reducibility

Let (X, R) and (X', R') be two computational problems, with $R : X \times Y$ and $R' : X' \times Y'$. We say that (X, R) is *conditionally* many-one reducible to (X', R') under *preconditions* c, which we denote $(X, R) \leq_m^c (X', R')$, if and only if $(\{x \in X \mid c(x)\}, R) \leq_m (X', R')$, where c is a Boolean function on X.

The practical relevance of a conditional reduction additionally depends on how commonly its preconditions are satisfied. It is worth noting that many-one reducibility is *transitive*, while conditional reducibility is not. However, the following holds:

$$(X, R) \leq_m^c (X', R') \quad \wedge \quad (X', R') \leq_m (X'', R'') \quad \implies \quad (X, R) \leq_m^c (X'', R'')$$

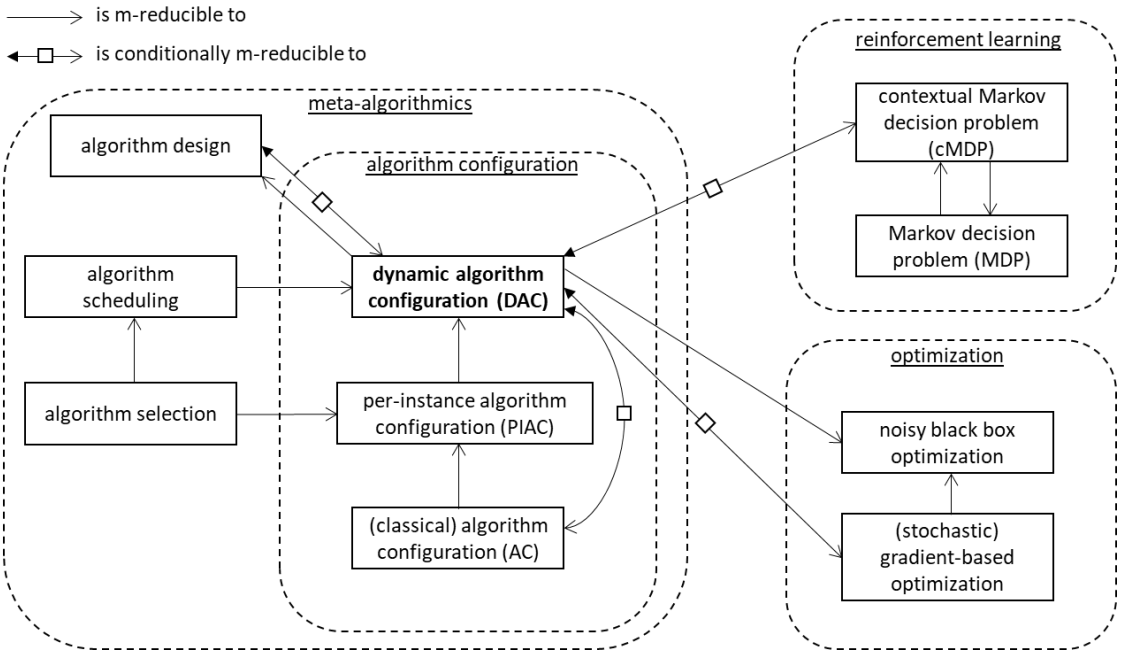


Figure 9: An overview of the reducibility relations between DAC and problems previously studied in the meta-algorithmics, reinforcement learning, and optimization communities; that we prove to exist in Section A.3. Note that arrows implied by the transitive/reflexive property of m-reducibility are not shown. Conditional reducibility indicates that a problem is only reducible to another, under specific conditions (i.e., not generally).

A.2 Reducibility Results

Figure 9 shows an overview of the reducibility relationships between DAC and all the other computational problems we discussed in the main text.

Reducibility to DAC: We observe that all problems in meta-algorithmics, discussed in Section 2.2.2, can be shown to be generally reducible to DAC. This suggests that research towards solving DAC in general, will indirectly find applications in solving many of these problems. Note that the *conditional* reduction from algorithm design corresponds to the "DAC powered PbO", discussed in Section 2.2.1, despite being conditional (i.e., not general), presents a highly practical approach to automating algorithm design.

Reducibility from DAC: We observe that DAC is generally reducible to "algorithm design" (as in Definition 7). However, this reduction is not practical since no general solvers for this problem are known. DAC is also generally reducible to "noisy black box optimization". While this reduction is more practical (see Section 4.2), a lot of information is lost in the process. Beyond these two problems, DAC is conditionally reducible. While many of these conditional reductions give rise to practical solution approaches (see Section 4), they are nonetheless limited both in terms of generality and the information they can exploit.

Conclusion: While a general DAC solver would allow us to solve many well-known computational problems, no such solver exists to date, and is therefore a research direction worth exploring. Note that various existing solvers can solve special cases of DAC, suggesting that another line of research would be to identify further special cases that can be solved more efficiently.

A.3 Reducibility Proofs

In this subsection, we formalize the reducibility relationships between DAC (Definition 3) and all problems discussed in the main text. Here, we first formally define each problem and then show reducibility by describing one possible reduction, i.e., defining `formulate` and `interpret` functions. Finally, we present a formal argument (*proof sketch*) for the correctness of each reduction.[14]

A.3.1 Algorithm Configuration

All discussed algorithm configuration variants were already defined in the main text:

- classical / per-distribution algorithm configuration (AC, Definition 1)
- per-instance algorithm configuration (PIAC, Definition 2)
- dynamic algorithm configuration (DAC, Definition 3)

In what follows, we formalize their relation.

14. For brevity, we generally proof (1) in Definition 5, but not corner case (2).

AC $\leq_m$ PIAC:

$\texttt{formulate}(\langle \mathcal{A}, \Theta, \mathcal{D}, c\rangle) = \langle \mathcal{A}, \Theta, \mathcal{D}, \Psi, c'\rangle$ with

- $\Psi = \{\psi_{\boldsymbol{\theta}} \,|\, \psi_{\boldsymbol{\theta}}(i) = \boldsymbol{\theta}, \forall \boldsymbol{\theta} \in \Theta\}$
- $c'(\psi_{\boldsymbol{\theta}}, i) = c(\boldsymbol{\theta}, i)$

$\texttt{interpret}(\langle \mathcal{A}, \Theta, \mathcal{D}, c\rangle, \psi_{\boldsymbol{\theta}^*}) = \boldsymbol{\theta}^*$

Proof Sketch: $\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\boldsymbol{\theta}, i)\right]$
By contradiction, assume $\boldsymbol{\theta}^* \notin \arg\min_{\boldsymbol{\theta}\in\Theta} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\boldsymbol{\theta}, i)\right]$. This implies that there exists $\boldsymbol{\theta}' : \mathbb{E}_{i\sim\mathcal{D}}\left[c(\boldsymbol{\theta}', i)\right] < \mathbb{E}_{i\sim\mathcal{D}}\left[c(\boldsymbol{\theta}^*, i)\right]$. Since $\boldsymbol{\theta}' \in \Theta$ and $c(\boldsymbol{\theta}, i) = c'(\psi_{\boldsymbol{\theta}}, i)$, there must exist $\psi_{\boldsymbol{\theta}'} \in \Psi$ having $\mathbb{E}_{i\sim\mathcal{D}}\left[c'(\psi_{\boldsymbol{\theta}'}, i)\right] < \mathbb{E}_{i\sim\mathcal{D}}\left[c'(\psi_{\boldsymbol{\theta}^*}, i)\right]$, contradicting $\psi_{\boldsymbol{\theta}^*} \in \arg\min_{\psi\in\Psi} \mathbb{E}_{i\sim\mathcal{D}}\left[c'(\psi, i)\right]$. □

PIAC $\leq_m$ DAC:

$\texttt{formulate}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Psi, c\rangle) = \langle \mathcal{A}', \Theta, \mathcal{D}, \Pi, c'\rangle$ with

- $\mathcal{A}'.\texttt{step}(s, i, \boldsymbol{\theta}) = \mathcal{A}(i, \boldsymbol{\theta})$ and $\mathcal{A}'.\texttt{init}(i) = \upsilon$ and $\mathcal{A}'.\texttt{is\_final}(s, i) \iff s \neq \upsilon$ (for some υ not being an output of $\mathcal{A}$, i.e., we perform exactly one step)
- $\Pi = \{\pi_\psi | \pi_\psi(s, i) = \psi(i), \forall \psi \in \Psi\}$
- $c'(\pi_\psi, i) = c(\psi, i)$

$\texttt{interpret}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Psi, c\rangle, \pi_{\psi^*}) = \psi^*$

Proof Sketch: $\psi^* \in \arg\min_{\psi\in\Psi} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\psi, i)\right]$.
By contradiction, assume $\psi^* \notin \arg\min_{\psi\in\Psi} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\psi, i)\right]$. This implies that there exists $\psi' : \mathbb{E}_{i\sim\mathcal{D}}\left[c(\psi', i)\right] < \mathbb{E}_{i\sim\mathcal{D}}\left[c(\psi^*, i)\right]$. Since $\psi' \in \Psi$, and $c(\psi, i) = c'(\pi_\psi, i)$, there must exist a $\pi_{\psi'} \in \Pi$ having $\mathbb{E}_{i\sim\mathcal{D}}\left[c'(\pi_{\psi'}, i)\right] < \mathbb{E}_{i\sim\mathcal{D}}\left[c'(\pi_{\psi^*}, i)\right]$, contradicting $\pi_{\psi^*} \in \arg\min_{\pi\in\Pi} \mathbb{E}_{i\sim\mathcal{D}}\left[c'(\pi, i)\right]$. □

DAC $\leq_m^c$ AC:

Preconditions: We assume to be given a parametric representation Λ of the policy space, i.e., $\Pi = \{\pi_{\boldsymbol{\lambda}} \,|\, \boldsymbol{\lambda} \in \Lambda\}$.

$\texttt{formulate}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c\rangle) = \langle \mathcal{A}', \Lambda, \mathcal{D}, c'\rangle$ with

- $\mathcal{A}'(i, \boldsymbol{\lambda}) = \mathcal{A}(i, \pi_{\boldsymbol{\lambda}})$.
- $c'(\boldsymbol{\lambda}, i) = c(\pi_{\boldsymbol{\lambda}}, i)$

$\texttt{interpret}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c\rangle, \boldsymbol{\lambda}^*) = \pi_{\boldsymbol{\lambda}^*}$

Proof Sketch: $\pi_{\boldsymbol{\lambda}^*} \in \arg\min_{\pi_{\boldsymbol{\lambda}}\in\Pi} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\pi_{\boldsymbol{\lambda}}, i)\right]$

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda}\in\Lambda} \mathbb{E}_{i\sim\mathcal{D}}\left[c'(\boldsymbol{\lambda}, i)\right] \implies$$
$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda}\in\Lambda} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\pi_{\boldsymbol{\lambda}}, i)\right] \implies$$
$$\pi_{\boldsymbol{\lambda}^*} \in \arg\min_{\pi_{\boldsymbol{\lambda}}\in\Pi} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\pi_{\boldsymbol{\lambda}}, i)\right]$$

□

A.3.2 Algorithm Design

We formalize algorithm design as in previous work (Adriaensen, 2018):

> **Definition 7: Algorithm Design**
>
> Let A_U be the space of all algorithms.[a] Given a preference relation over $\preceq^b$ over A_U, find a $a^* \in A_U : a^* \not\prec a, \forall a \in A_U$.
>
> a. A_U is a universal set containing "any procedure that solves some problem". Further formalization of this notion is hindered by the lack of a generally accepted, formal definition of "an algorithm".
> b. $\preceq$ is assumed to be a preorder, i.e., a binary relation that is reflexive and transitive.

algorithm design $\leq_m^c$ DAC:

Preconditions: We assume we are given `init`, `step`, `is_final`, and a set of Π sub-routines such that algorithms $a_\pi \in A_\Pi \subset A_U$ that can be decompose in $\langle$`init`, `step`, `is_final`, $\pi\rangle$ as in Algorithm 1 are at least as preferable as any other: $\forall a_\pi \in A_\Pi : a \preceq a_\pi, \forall a \in A_U \setminus A_\Pi$.

$\texttt{formulate}(\langle \preceq, \texttt{init}, \texttt{step}, \texttt{is\_final}, \Pi\rangle) = \langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c\rangle$ with

- $\mathcal{A}.\texttt{step} = \texttt{step}$ and $\mathcal{A}.\texttt{init} = \texttt{init}$ and $\mathcal{A}.\texttt{is\_final} = \texttt{is\_final}$
- Any choice of $\mathcal{D}$ and c such that $a_{\pi'} \prec a_\pi \implies \mathbb{E}_{i\sim\mathcal{D}}[c(\pi, i)] < \mathbb{E}_{i\sim\mathcal{D}}[c(\pi', i)]$. This can always be achieved, as a c that is solely a function of π can impose an arbitrary total order on Π and therefore also one consistent with $\preceq$.

$\texttt{interpret}(\langle \preceq, \texttt{init}, \texttt{step}, \texttt{is\_final}, \Pi\rangle, \pi^*) = a_{\pi^*}$

Proof Sketch: $a_{\pi^*} \not\prec a, \forall a \in A_U$
By contradiction, assume there exists $a' \in A_U : a' \succ a_{\pi^*}$. Given our pre-condition, we have $a' = a_{\pi'} \in A_\Pi$. From our choice of c follows that $\mathbb{E}_{i\sim\mathcal{D}}[c(\pi', i)] < \mathbb{E}_{i\sim\mathcal{D}}[c(\pi^*, i)]$, contradicting $\pi^* \in \arg\min_{\pi\in\Pi} \mathbb{E}_{i\sim\mathcal{D}}[c(\pi, i)]$. □

DAC $\leq_m$ algorithm design:

$\texttt{formulate}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c\rangle) = \preceq$
satisfying

1. $\forall \pi \in \Pi : a \prec \pi, \forall a \in A_U \setminus \Pi$ and
2. $\forall \pi, \pi' \in \Pi : \mathbb{E}_{i\sim\mathcal{D}}[c(\pi, i)] < \mathbb{E}_{i\sim\mathcal{D}}[c(\pi', i)] \implies \pi' \prec \pi$.

$\texttt{interpret}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c\rangle, a^*) = a^*$

Proof Sketch: $a^* \in \arg\min_{\pi\in\Pi} \mathbb{E}_{i\sim\mathcal{D}}[c(\pi, i)]$
By contradiction, assume $a^* \notin \arg\min_{\pi\in\Pi} \mathbb{E}_{i\sim\mathcal{D}}[c(\pi, i)]$. First, note that $a^* \in A_U \setminus \Pi$ and (1) would contradict $a^* \in A_U : a^* \not\prec a, \forall a \in A_U$. This implies there exists $\pi' \in \Pi : \mathbb{E}_{i\sim\mathcal{D}}[c(\pi', i)] < \mathbb{E}_{i\sim\mathcal{D}}[c(a^*, i)]$, implying $a^* \prec \pi'$ by (2) and contradicting $a^* \in A_U : a^* \not\prec a, \forall a \in A_U$. □

A.3.3 Algorithm Selection

For algorithm selection, we adopt the classical definition by Rice (1976):[15]

> **Definition 8: Algorithm Selection**
>
> Given $\langle A, I, c \rangle$:
>
> – A finite set A of target algorithms
>
> – A target problem space I
>
> – A cost metric $c : A \times I \to \mathbb{R}$ assessing the cost of solving $i \in I$ using $a \in A$.[a]
>
> Find a selection mapping $S^* : I \to A$ satisfying $S^*(i) \in \arg\min_{a \in A} c(a, i), \forall i \in I$.
>
> a. The original definition uses a performance metric p (to be maximized).

The reducability algorithm selection to DAC follows by transitive property from its reducability to PIAC.

algorithm selection $\leq_m$ PIAC:

$\texttt{formulate}(\langle A, I, c \rangle) = \langle \mathcal{A}', \Theta, \mathcal{D}, \Psi, c' \rangle$ with

- $\mathcal{A}'(i, k) = a_k(i)$
- $\Theta = \{k \mid a_k \in A\}$ (single categorical parameter)
- $\mathcal{D} = U(I)$
- $\Psi : I \to \Theta$ (unconstrained)
- $c'(\psi, i) = c(\psi(i), i)$

$\texttt{interpret}(\langle A, I, c \rangle, \psi^*) = S^*$ with $S^*(i) = a_{\psi^*(i)}$

Proof Sketch: $S^*(i) \in \arg\min_{a \in A} c(a, i), \forall i \in I$

By contradiction, assume $\exists j \in I : S^*(j) \notin \arg\min_{a \in A} c(a, j)$. This implies there exists $S' : S'(j) \in \arg\min_{a \in A} c(a, j) \wedge S'(i') = S(i'), \forall i' \in I \setminus \{j\}$. Since Ψ is unconstrained every selection mapping S has its corresponding $\psi \in \Psi : S(i) = a_{\psi(i)}$ and therefore $\exists\, \psi' \in \Psi : c(\psi'(j), j) < c(\psi^*(j), j) \wedge c(\psi'(i'), i') = c(\psi^*(i), i'), \forall i' \in I \setminus \{j\}$. From $c'(\psi, i) = c(\psi(i), i)$ and $\mathcal{D}(j) > 0$ follows that $\mathbb{E}_{i \sim \mathcal{D}}[c'(\psi', i)] < \mathbb{E}_{i \sim \mathcal{D}}[c'(\psi^*, i)]$, contradicting $\psi^* \in \arg\min_{\psi \in \Psi} \mathbb{E}_{i \sim \mathcal{D}}[c'(\psi, i)]$. □

A.3.4 Algorithm Scheduling

We define a variant of algorithm scheduling that considers allocating a fixed time budget to a finite set of target algorithms in an instance-aware and dynamic fashion:

15. Rice (1976) defines many different more general variants of the problem. However, we will restrict ourselves to a canonical variant, i.e., selecting the best algorithm, per-instance, from finite alternatives, without constraints on the selection mappings.

Definition 9: Algorithm Scheduling

Given $\langle A, B, I, \Delta, c\rangle$:

– A finite set A of step-wise executable target algorithms such that the execution of the k^{th} algorithm $a_k \in A$ can be decomposed as a consecutive application of a sub-routine $a_k.\texttt{tstep}$ such that the state of algorithm a_k when solving a problem instance i, after t time steps, is given by $s_{k,i,t} = a_k.\texttt{tstep}(s_{k,i,t-1})$ with $s_{k,i,0} = i$.

– A finite budget B of time steps to be allocated to algorithms in A.

– A target problem space I

– A space of *dynamic scheduling policies* $\delta \in \Delta$ with $\delta : \mathcal{S}^{|A|} \times I \times \mathbb{N} \to A$ choosing which algorithm $a_k \in A$ to resume executing in the next time step, as a function of the total time $T \in \mathbb{N}$ elapsed thus far, the instance $i \in I$ being solved and the vector of states s_{m,i,t_m} of each algorithm $a_m \in A$.

– A cost metric $c : A \times I \times \mathbb{N} \to \mathbb{R}$ assessing the cost of solving $i \in I$ using $a \in A$ for $t \in \mathbb{N}$ time steps.

Find a dynamic scheduling policy $\delta^* \in \arg\min_{\delta\in\Delta} \sum_{i\in I}(\min_{a_k\in A} c(a_k, i, t_k^{B,\delta,i}))$ where $t_k^{T,\delta,i}$ is the total time allocated to a_k by δ after T scheduling steps on instance i, and is given by

$$t_k^{T,\delta,i} = \begin{cases} 0 & T = 0 \\ t_k^{T-1,\delta,i} & T > 0 \quad \wedge \quad a_k \neq \delta(\boldsymbol{s}^{T-1,\delta,i}, i, T-1) \\ t_k^{T-1,\delta,i} + 1 & T > 0 \quad \wedge \quad a_k = \delta(\boldsymbol{s}^{T-1,\delta,i}, i, T-1) \end{cases}$$

with

$$s_k^{T,\delta,i} = \begin{cases} i & T = 0 \\ s_k^{T-1,\delta,i} & T > 0 \quad \wedge \quad a_k \neq \delta(\boldsymbol{s}^{T-1,\delta,i}, i, T-1) \\ a_k.\texttt{tstep}(s_k^{T-1,\delta,i}) & T > 0 \quad \wedge \quad a_k = \delta(\boldsymbol{s}^{T-1,\delta,i}, i, T-1) \end{cases}$$

algorithm scheduling $\leq_m$ DAC:

$\texttt{formulate}(\langle A, B, I, \Delta, c\rangle) = \langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c'\rangle$ with

- $\mathcal{A}.\texttt{step}((\boldsymbol{s}, i, T), i, \theta) = (\boldsymbol{s}', i, T+1)$ with $s'_k = \begin{cases} s_k & \theta \neq k \\ a_k.\texttt{tstep}(s_k) & \theta = k \end{cases}$
  $\mathcal{A}.\texttt{init}(i) = (\boldsymbol{s}, i, 0)$ with $s_k = i$
  $\mathcal{A}.\texttt{is\_final}((\boldsymbol{s}, i, T), i) \Leftrightarrow T = B.$
- $\Theta = \{k \mid a_k \in A\}$
- $\mathcal{D} = U(I)$
- $\Pi = \{\pi_\delta \mid \delta \in \Delta\}$ with $\pi_\delta((\boldsymbol{s}, i, T), i) = k \Leftrightarrow \delta(\boldsymbol{s}, i, T) = a_k$
- $c'(\pi_\delta, i) = \min_{a_k\in A} c(a_k, i, t_k^{B,\delta,i})$

$\texttt{interpret}(\langle A, B, I, \Delta, c\rangle, \pi_{\delta^*}) = \delta^*$

Proof Sketch: $\delta^* \in \arg\min_{\delta\in\Delta} \sum_{i\in I}(\min_{a_k\in A} c(a_k, i, t_k^{B,\delta,i}))$

$$\pi_{\delta^*} \in \arg\min_{\pi_\delta\in\Pi} \mathbb{E}_{i\sim\mathcal{D}}\left[c'(\pi_\delta, i)\right] \implies$$
$$\delta^* \in \arg\min_{\delta\in\Delta} \mathbb{E}_{i\sim\mathcal{D}}\left[\min_{a_k\in A} c(a_k, i, t_k^{B,\delta,i})\right] \implies$$
$$\delta^* \in \arg\min_{\delta\in\Delta} \frac{1}{|I|}\sum_{i\in I}\left[\min_{a_k\in A} c(a_k, i, t_k^{B,\delta,i})\right] \implies$$
$$\delta^* \in \arg\min_{\delta\in\Delta} \sum_{i\in I}(\min_{a_k\in A} c(a_k, i, t_k^{B,\delta,i}))$$

□

A.3.5 Reinforcement Learning

In Section 4.1, we discussed the relation between DAC and the

Definition 10: Markov Decision Problem (MDP)

Given $\langle S, A, T, R\rangle$:

- A state space S.
- An action space A
- A transition function $T : S \times A \to S$
- A reward function $R : S \times A \to \mathbb{R}$

Find a policy $\pi : S \to A$ satisfying $\pi^*(s) \in \arg\max_{a\in A} R(s, a) + V^*(T(s, a))$ with V^* being the optimal value-state function, i.e., $V^*(s) = \max_{a\in A} R(s, a) + V^*(T(s, a))$.

Note that we will restrict ourselves to episodic MDPs, where we have absorbing states $S_{\mathcal{H}} = \{s \,|\, T(s, a) = s, \forall a \in A\}$ having $R(s, a) = 0, \forall s \in S_{\mathcal{H}}$ that are reached within an arbitrarily large, but finite horizon $\mathcal{H}$ and therefore $V^*(s) = \max_\pi \sum_{t=0}^{\mathcal{H}-1} R(s_{\pi,t}, \pi(s_{\pi,t}))$ where $s_{\pi,t} = T(s_{\pi,t-1}, \pi(s_{\pi,t-1}))$ is the t^{th} state encountered when following π starting in $s_{\pi,0} = s$.

Since standard RL methods are not instance-aware, Biedenkapp et al. (2020) proposed to model DAC as a

Definition 11: Contextual Markov Decision Problem (cMDP, Hallak et al., 2015)

Given $\langle \mathcal{C}, S, A, \mathcal{M} \rangle$:

- A context space $\mathcal{C}$
- A shared state space S
- A shared action space A
- A function $\mathcal{M}$ mapping any $c \in \mathcal{C}$ to an MDP $\mathcal{M}(c) = \langle S, A, T_c, R_c \rangle$ with
  - a context-dependent transition function $T_c : S \times A \to S$
  - a context-dependent reward function $R_c : S \times A \to \mathbb{R}$

Find a policy $\pi : S \times \mathcal{C} \to A$ satisfying $\pi^*(s, c) \in \arg\max_{a \in A} R_c(s, a) + V_c^*(T_c(s, a))$ with $V_c^*(s) = \max_{a \in A} R_c(s, a) + V_c^*(T_c(s, a))$.

Please remark that we assume the context to be observable and our objective to be finding an optimal *context-dependent* policy. We will also assume MDP $\mathcal{M}(i)$ to be episodic. As a consequence, this formulation is m-equivalent to that of an ordinary episodic MDP. The cMDP formulation is nonetheless interesting in that it can capture various aspects of DAC abstracted in the MDP reduction: DAC $\leq_m$ MDP $\leq_m$ cMDP.

DAC $\leq_m^c$ cMDP:

Preconditions:

1. The cost function c is step-wise decomposable, i.e., we are given functions $\langle \mathrm{c}_{\text{init}}, \mathrm{c}_{\text{step}} \rangle$, such that

$$c(\pi, i) = \mathrm{c}_{\text{init}}(i) + \sum_{t=0}^{T-1} \mathrm{c}_{\text{step}}(s_t, i, \pi(s_t, i))$$

   where

$$s_0 = \texttt{init}(i) \quad \wedge \quad s_t = \texttt{step}(s_{t-1}, i, \pi(s_{t-1}, i)) \quad \wedge \quad \texttt{is\_final}(s_t, i) \Leftrightarrow t = T$$

2. The policy space is unconstrained, i.e., $\Pi = \{\pi \mid \pi(s, i) \in \Theta, \ \forall\, s \in \mathcal{S} \wedge i \in I\}$

$\texttt{formulate}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c \rangle) = \langle \mathcal{C}, S, A, \mathcal{M} \rangle$ with

- $\mathcal{C} = I$ the domain of $\mathcal{D}$.
- $S = \mathcal{S}$ the set of algorithm states.
- $A = \Theta$
- $\mathcal{M}(i) = \langle S, A, T_i, R_i \rangle$ with

- $T_i(s, \boldsymbol{\theta}) = \begin{cases} s & \texttt{is\_final}(s, i) \\ \texttt{step}(s, i, \boldsymbol{\theta}) & \neg\, \texttt{is\_final}(s, i) \end{cases}$
- $R_i(s, \boldsymbol{\theta}) = \begin{cases} 0 & \texttt{is\_final}(s, i) \\ \mathrm{c}_{\mathrm{step}}(s, i, \boldsymbol{\theta}) & \neg\, \texttt{is\_final}(s, i) \end{cases}$

$\texttt{interpret}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c \rangle, \pi^*) = \pi^*$

Proof Sketch: $\pi^* \in \arg\max_\pi \mathbb{E}_{i \sim \mathcal{D}}\, c(\pi, i)$

We first note that since $S = \mathcal{S} \wedge A = \Theta \wedge \mathcal{C} = I$ and given precondition (2), it follows that both problems have the exact same policy space Π. It remains to show that optimality in the resulting cMDP implies optimality in the original DAC:

$$\pi^*(s, i) \in \arg\max_{\boldsymbol{\theta} \in \Theta} R_i(s, \boldsymbol{\theta}) + V_i^*(T_i(s, \boldsymbol{\theta})) \quad (\forall s \in S, \forall i \in I) \implies$$
$$\pi^*(\texttt{init}(i), i) \in \arg\max_{\boldsymbol{\theta} \in \Theta} R_i(\texttt{init}(i), \boldsymbol{\theta}) + V_i^*(T_i(\texttt{init}(i), \boldsymbol{\theta})) \quad (\forall i \in I) \implies$$
$$\pi^* \in \arg\max_\pi \mathbb{E}_{i \sim \mathcal{D}} R_i(\texttt{init}(i), \pi(\texttt{init}(i), i)) + V_i^*(T_i(\texttt{init}(i), \pi(\texttt{init}(i), i)))$$

Since each $\mathcal{M}(i)$ is episodic, this objective can be rewritten as:

$$\pi^* \in \arg\max_\pi \mathbb{E}_{i \sim \mathcal{D}} \sum_{t=0}^{\mathcal{H}-1} R_i(s_{\pi,t,i}, \pi(s_{\pi,t,i})) \text{ where } s_{\pi,t,i} = \begin{cases} \texttt{init}(i) & t = 0 \\ T_i(s_{\pi,t-1,i}, \pi(s_{\pi,t-1,i})) & t > 0 \end{cases}$$
$$\pi^* \in \arg\max_\pi \mathbb{E}_{i \sim \mathcal{D}} \sum_{t=0}^{T-1} \mathrm{c}_{\mathrm{step}}(s_{\pi,t,i}, i, \pi(s_{\pi,t,i})) \text{ where } s_{\pi,t,i} = \begin{cases} \texttt{init}(i) & t = 0 \\ \texttt{step}(s_{\pi,t-1,i}, i, \pi(s_{\pi,t-1,i})) & t > 0 \end{cases}$$
$$\pi^* \in \arg\max_\pi \mathbb{E}_{i \sim \mathcal{D}}\, c(\pi, i)$$

□

It is worth noting that the optimality of π^* does not depend on $\mathcal{D}$, $\texttt{init}$, or $\mathrm{c}_{\mathrm{init}}$.
Conditional reducability to an ordinary MDP follows from the transitive property and

cMDP $\leq_m$ MDP:

$\texttt{formulate}(\langle \mathcal{C}, S, A, \mathcal{M} \rangle) = \langle S', A, T, R \rangle$ with

- $S' = S \times \mathcal{C}$
- $T((s, c), a) = T_c(s, a)$
- $R((s, c), a) = R_c(s, a)$

where $\mathcal{M}(c) = \langle S, A, T_c, R_c \rangle$.

$\texttt{interpret}(\langle \mathcal{C}, S, A, \mathcal{M} \rangle, \pi^*) = \pi'^*$ with $\pi'^*(s, c) = \pi^*((s, c))$

Proof Sketch: $\pi'^*(s,c) \in \arg\max_{a \in A} R_c(s,a) + V_c^*(s)$

$$\pi^*((s,c)) \in \arg\max_{a \in A} R((s,c),a) + V^*(T((s,c),a)) \implies$$
$$\pi'^*(s,c) \in \arg\max_{a \in A} R_c(s,a) + V^*(T_c(s,a))$$

It remains to show that $V^*((s,c)) = V_c^*(s)$. We can proof this by induction on the maximum number of steps $\mathcal{H}_{s,c}$ before reaching an absorbing state following any policy starting from state s in context c. The base case $\mathcal{H}_{s,c} = 0$ follows from $T((s,c),a) = (s,c) \iff T_c(s,a) = s$. Now, for the recursive case, assuming this holds for $\mathcal{H}_{s,c} \leq n-1$, it also holds for $\mathcal{H}_{s,c} = n$

$$\begin{aligned} V^*((s,c)) &= \max_{a \in A} R((s,c),a) + V^*(T((s,c),a)) \\ &= \max_{a \in A} R_c(s,a) + V^*(T_c(s,a)) \\ &= \max_{a \in A} R_c(s,a) + V_c^*(T_c(s,a)) = V_c^*(s) \end{aligned}$$

since $\mathcal{H}_{T_c(s,a),c} \leq n-1$. □

and both are m-equivalent since

MDP $\leq_m$ cMDP:

`formulate`$(\langle S, A, T, R \rangle) = \langle \mathcal{C}, S, A, \mathcal{M} \rangle$ with

- $\mathcal{C} = \{0\}$
- $T_0(s,a) = T(s,a)$
- $R_0(s,a) = R(s,a)$

where $\mathcal{M}(0) = \langle S, A, T_0, R_0 \rangle$.

`interpret`$(\langle S, A, T, R \rangle, \pi^*) = \pi'^*$ with $\pi'^*(s) = \pi^*(s,0)$

Proof Sketch: $\pi'^*(s) \in \arg\max_{a \in A} R(s,a) + V^*(s)$

$$\pi^*(s,0) \in \arg\max_{a \in A} R_0(s,a) + V_0^*(T_0(s,a)) \implies$$
$$\pi'^*(s) \in \arg\max_{a \in A} R(s,a) + V_0^*(T(s,a))$$

It remains to show that $V^*(s) = V_0^*(s)$. We can proof this by induction on the maximum number of steps $\mathcal{H}_s$ before reaching an absorbing state following any policy starting from state s. The base case $\mathcal{H}_s = 0$ follows from $T(s,a) = s \Leftrightarrow T_0(s,a) = s$. Now the recursive case, assuming this holds for $\mathcal{H}_s \leq n-1$, it also holds for $\mathcal{H}_s = n$

$$\begin{aligned} V^*(s) &= \max_{a \in A} R(s,a) + V^*(T(s,a)) \\ &= \max_{a \in A} R_0(s,a) + V^*(T_0(s,a)) \\ &= \max_{a \in A} R_0(s,a) + V_0^*(T_0(s,a)) = V_0^*(s) \end{aligned}$$

since $\mathcal{H}_{T_0(s,a)} \leq n-1$. □

A.3.6 Optimization

In Section 4.2, we discussed the relation between DAC and

Definition 12: Noisy Black Box Optimization

Given $\langle X, e \rangle$:

- A search space X
- A noisy evaluation sub-routine e

Find a $x^* \in \arg\min_{x \in X} \mathbb{E}[e(x)]$.

DAC $\leq_m$ noisy black box optimization:

$\texttt{formulate}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c \rangle) = \langle \Pi, e \rangle$ where $e(\pi) = c(\pi, i)$ with $i \sim \mathcal{D}$.

$\texttt{interpret}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c \rangle), \pi^*) = \pi^*$

Proof Sketch: $\pi^* \in \arg\min_{\pi \in \Pi} \mathbb{E}_{i \sim \mathcal{D}}[c(\pi, i)]$
We have $\pi^* \in \arg\min_{\pi \in \Pi} \mathbb{E}[e(\pi)]$, since π^* is a solution for noisy black box optimization problem. Since $e(\pi) = c(\pi, i)$ and $i \sim \mathcal{D}$, this is equivalent to $\pi^* \in \arg\min_{\pi \in \Pi} \mathbb{E}_{i \sim \mathcal{D}}[c(\pi, i)]$. □

Also in Section 4.2, we discussed the possibility of solving DAC using

Definition 13: Stochastic Gradient-Based Optimization

Given $\langle X, e, e' \rangle$:

- X and e as in Definition 12.
- A stochastic differentiation sub-routine e' satisfying $\mathbb{E}[e'(x)] = \mathbb{E}[\frac{\partial e(x)}{\partial x}]$.

Find a $x^* \in \arg\min_{x \in X} \mathbb{E}[e(x)]$.

DAC $\leq_m^c$ stochastic gradient-based optimization:

Preconditions:

1. We assume to be given a parametric representation Λ of the policy space, i.e., $\Pi = \{\pi_{\boldsymbol{\lambda}} \mid \boldsymbol{\lambda} \in \Lambda\}$.
2. We assume $c(\pi_{\boldsymbol{\lambda}}, i)$ to be piece-wise differentiable w.r.t. $\boldsymbol{\lambda}$ and a sub-routine for calculating $\frac{\partial c(\pi_{\boldsymbol{\lambda}}, i)}{\partial \boldsymbol{\lambda}}$ to be given.

$\texttt{formulate}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c\rangle) = \langle \Lambda, e, e'\rangle$ with

- $e(\boldsymbol{\lambda}) = c(\pi_{\boldsymbol{\lambda}}, i)$ with $i \sim \mathcal{D}$.
- $e'(\boldsymbol{\lambda}) = \frac{\partial c(\pi_{\boldsymbol{\lambda}}, i)}{\partial \boldsymbol{\lambda}}$ with $i \sim \mathcal{D}$.

$\texttt{interpret}(\langle \mathcal{A}, \Theta, \mathcal{D}, \Pi, c\rangle, \boldsymbol{\lambda}^*) = \pi_{\boldsymbol{\lambda}^*}$

Proof Sketch: $\pi_{\boldsymbol{\lambda}^*} \in \arg\min_{\pi_{\boldsymbol{\lambda}} \in \Pi} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\pi_{\boldsymbol{\lambda}}, i)\right]$
We have $\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda}\in\Lambda} \mathbb{E}[e(\boldsymbol{\lambda})]$, since $\boldsymbol{\lambda}^*$ is a solution for black box optimization problem. Since $e(\boldsymbol{\lambda}) = c(\pi_{\boldsymbol{\lambda}}, i)$ and $i \sim \mathcal{D}$, this is equivalent to $\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda}\in\Lambda} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\pi_{\boldsymbol{\lambda}}, i)\right]$. Substituting every $\boldsymbol{\lambda}$ by its corresponding policy $\pi_{\boldsymbol{\lambda}}$, we get $\pi_{\boldsymbol{\lambda}^*} \in \arg\min_{\pi_{\boldsymbol{\lambda}}\in\Pi} \mathbb{E}_{i\sim\mathcal{D}}\left[c(\pi_{\boldsymbol{\lambda}}, i)\right]$
□

At first sight, precondition (2) may seem very strong. In what follows, we show a sufficient condition that is arguably not so strong.

Sufficient Conditions: Next to the precondition (1), we assume

3. to be given a parametric representation of the state space.
4. the cost function c to be step-wise decomposable in $\langle \mathrm{c}_{\text{init}}, \mathrm{c}_{\text{step}}\rangle$ such that

$$c(\pi_{\boldsymbol{\lambda}}, i) = \mathrm{c}_{\text{init}}(i) + \sum_{t=0}^{T-1} \mathrm{c}_{\text{step}}(s_t, i, \pi_{\boldsymbol{\lambda}}(s_t, i))$$

where

$$s_0 = \texttt{init}(i) \quad \wedge \quad s_t = \texttt{step}(s_{t-1}, i, \pi_{\boldsymbol{\lambda}}(s_{t-1}, i)) \quad \wedge \quad \texttt{is\_final}(s_t, i) \Leftrightarrow t = T$$

5. to be given sub-routines for calculating the following partial derivatives: $\frac{\partial \pi_{\boldsymbol{\lambda}}(s,i)}{\partial \boldsymbol{\lambda}}$, $\frac{\partial\, \mathrm{c}_{\text{step}}(s,i,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$, $\frac{\partial\, \mathrm{c}_{\text{step}}(s,i,\boldsymbol{\theta})}{\partial s}$, $\frac{\partial\, \texttt{step}(s,i,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$, and $\frac{\partial\, \texttt{step}(s,i,\boldsymbol{\theta})}{\partial s}$.

Proof Sketch: Under these conditions precondition 2 also holds.
The piece-wise derivative of c w.r.t. $\boldsymbol{\lambda}$ can be calculated as follows

$$\begin{aligned}
\frac{\partial c(\pi_{\boldsymbol{\lambda}}, i)}{\partial \boldsymbol{\lambda}} &= \frac{\partial \left(\mathrm{c}_{\text{init}}(i) + \sum_{t=0}^{T-1} \mathrm{c}_{\text{step}}(s_t, i, \pi_{\boldsymbol{\lambda}}(s_t, i))\right)}{\partial \boldsymbol{\lambda}} \\
&= \frac{\partial\, \mathrm{c}_{\text{init}}(i)}{\partial \boldsymbol{\lambda}} + \sum_{t=0}^{T-1} \frac{\partial\, \mathrm{c}_{\text{step}}(s_t, i, \pi_{\boldsymbol{\lambda}}(s_t, i))}{\partial \boldsymbol{\lambda}} \\
&= \sum_{t=0}^{T-1} \frac{\partial\, \mathrm{c}_{\text{step}}(s_t, i, \pi_{\boldsymbol{\lambda}}(s_t, i))}{\partial \boldsymbol{\lambda}}
\end{aligned}$$

where

$$\frac{\partial\, \mathrm{c}_{\text{step}}(s_t, i, \pi_{\boldsymbol{\lambda}}(s_t, i))}{\partial \boldsymbol{\lambda}} = \frac{\partial\, \mathrm{c}_{\text{step}}(s_t, i, \pi_{\boldsymbol{\lambda}}(s_t, i))}{\partial \pi_{\boldsymbol{\lambda}}(s_t, i)} \cdot \frac{\partial \pi_{\boldsymbol{\lambda}}(s_t, i)}{\partial \boldsymbol{\lambda}} + \frac{\partial s_t}{\partial \boldsymbol{\lambda}} \cdot \left(\frac{\partial\, \mathrm{c}_{\text{step}}(s_t, i, \pi_{\boldsymbol{\lambda}}(s_t, i))}{\partial s_t} + \frac{\partial\, \mathrm{c}_{\text{step}}(s_t, i, \pi_{\boldsymbol{\lambda}}(s_t, i))}{\partial \pi_{\boldsymbol{\lambda}}(s_t, i)} \cdot \frac{\partial \pi_{\boldsymbol{\lambda}}(s_t, i)}{\partial s_t} \right)$$

with

$$\frac{\partial s_t}{\partial \boldsymbol{\lambda}} = \frac{\partial\, \texttt{step}(s_{t-1}, i, \pi_{\boldsymbol{\lambda}}(s_{t-1}, i))}{\pi_{\boldsymbol{\lambda}}(s_{t-1}, i)} \cdot \frac{\partial \pi_{\boldsymbol{\lambda}}(s_{t-1}, i)}{\partial \boldsymbol{\lambda}} + \frac{\partial s_{t-1}}{\partial \boldsymbol{\lambda}} \cdot \left(\frac{\partial\, \texttt{step}(s_{t-1}, i, \pi_{\boldsymbol{\lambda}}(s_{t-1}, i))}{\partial s_{t-1}} + \frac{\partial\, \texttt{step}(s_{t-1}, i, \pi_{\boldsymbol{\lambda}}(s_{t-1}, i))}{\partial \pi_{\boldsymbol{\lambda}}(s_{t-1}, i)} \cdot \frac{\partial \pi_{\boldsymbol{\lambda}}(s_{t-1}, i)}{\partial s_{t-1}} \right)$$

if $t > 0$ and $\frac{\partial s_0}{\partial \boldsymbol{\lambda}} = 0$

Note that we can reuse the quantities calculated in the previous step, resulting in a procedure known as *forward-mode* differentiation. While we will not derive the formulas here, $\frac{\partial c(\pi_{\boldsymbol{\lambda}}, i)}{\partial \boldsymbol{\lambda}}$ can also be calculated using *reverse-mode* differentiation, a procedure also known as *backpropagation* in the context of neural networks. □

Appendix C

Appendix for Learning Step-Size Adaptation in CMA-ES

Supplementary Material for: Learning Step-Size Adaptation in CMA-ES

Gresa Shala[1], André Biedenkapp[1], Noor Awad[1], Steven Adriaensen[1], Marius Lindauer[2], and Frank Hutter[1,3]

[1] University of Freiburg, Germany
[2] Leibniz University Hannover, Germany
[3] Bosch Center for Artificial Intelligence

A Available Software and Trained Policies

To enable other researchers to use our code, as well as trained models both are publicly available at https://github.com/automl/LTO-CMA. All scripts used to generate and plot our results, as well as the logged data are provided in the repository. Further, we provide examples of how to use our trained policy networks with the python version of CMA-ES (pycma) in version 2.7.0.

B Influence of Different Reward Scales

A drawback of prior methods of dynamic algorithm configuration through model-free RL is the need to learn the value function to find a well performing policy. With such value-based approaches learning across environments with very different reward scales is more challenging than with guided policy search, since the value function and therefore the policy can quickly be dominated by an environment with a large reward scale. Guided policy search on the other hand does not learn a value function to determine a well performing policy. GPS rather makes use of the reward signal to determine in which direction a better final reward can be achieved. On each condition it optimizes trajectory controllers individually with respect to reward and then learns policies that are similar to those teaching trajectories. Thus the learning policy is not influenced by different reward scales and simply learns to imitate teachers that were optimized for each function individually.

C Experimental Setup

We evaluated our approach on a compute cluster with nodes equipped with two Intel Xeon Gold 6242 32-core CPUs, 20 MB cache and and 188GB (shared) RAM running Ubuntu 18.04 LTS 64 bit.

Function Specific Policy For each of the 10 BBOB functions, we trained the policies on a set of 18 conditions consisting of the same function, but with different initialization values for the mean and step-size of CMA-ES.

Function Class Specific Policy For each of the 10 BBOB functions, we trained the policies on a set of 48 conditions consisting of the same function, but with different dimensionality(5D, 10D, 15D, 20D, 25D, 30D), initialization values for the mean and step-size of CMA-ES.

General Policy for BBOB We trained the policy on a set of 80 conditions consisting of 8 conditions for each function with the same dimensionality (10D), but with different initialization values for the mean and step-size of CMA-ES.

Used Metric We evaluated each method for n runs. To compare the resulting performance of using our learned policy to that of the handcrafted baseline, we compared the final performance of each of the 25 runs of our method to that of the handcrafted baseline. We count how often our method outperforms the baseline over all comparisons which gives us a probability of our policy π outperforming the baseline CSA as

$$p(\pi < CSA) = \frac{\sum_i^n \sum_j^n \mathbb{1}_{\pi_i < CSA_j}}{n^2} \tag{1}$$

where $\mathbb{1}_{\pi_i < CSA_j}$ is the indicator function showing if our policy resulted in a lower final objective value than the baseline when comparing runs i and j.

Statistical Significance: The performance metric used (see Equation 1) can easily be interpreted statistically as there exists a correspondence between $p(\pi < CSA)$ and the 'sum of ranks' statistic (W) used in the Wilcoxon rank-sum test, i.e. $p(\pi < CSA) = \frac{W_{\max} - W}{W_{\max} - W_{\min}}$ with $W_{\max} = \frac{n(3n+1)}{2}$ and $W_{\min} = \frac{n(n+1)}{2}$. As such, we can use critical values of the test statistic W [54] to derive critical values for $p(\pi < CSA)$, allowing us to draw conclusions about the significance of our results simply by comparing the value of this metric to a fixed threshold. Table C1 gives these thresholds for different common confidence levels α with $n = 25$.

p-value $< \alpha =$	**0.1**	**0.05**	**0.025**	**0.01**	**0.005**	**0.001**
$\mathbf{W} <$	570	552	536	517	505	480
$\mathbf{p}(\pi < \mathbf{CSA}) \geq$	0.61	0.64	0.67	0.70	0.72	0.76

Table C1: Critical values in terms of W ("sum of ranks" statistic) and $p(\pi < CSA)$ ("probability of outperforming" metric) for a single-sided Wilcoxon's rank-sum test with null-hypothesis: "CMA-ES with CSA performs at least as good as with our learned controller π" at different confidence levels α and $n = 25$.

	Sampling Rate									
	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
BentCigar	0.00	0.00	0.19	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
Discus	0.00	0.00	0.00	**1.00**	0.47	0.06	0.00	0.00	0.00	0.00
Ellipsoid	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	**1.00**	0.00	0.00
Katsuura	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rastrigin	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rosenbrock	**1.00**	0.60	**1.00**	**1.00**	**1.00**	0.00	0.19	**1.00**	**1.00**	0.00
Schaffers	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
Schwefel	**1.00**	0.00	0.00	**1.00**	0.00	**1.00**	0.00	**1.00**	0.00	**1.00**
Sphere	0.00	**1.00**	0.00	**1.00**	0.00	0.00	**1.00**	0.00	0.00	0.00
Weierstrass	0.00	**1.00**	**1.00**	**1.00**	0.00	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Average	0.40	0.46	0.42	**1.00**	0.35	0.41	0.42	0.60	0.40	0.40

Table D1: Probability of our approach outperforming CSA, in terms of AUC, for different dimensions on 10 BBOB functions with different sampling rates.

D Additional Analysis using Area Under the Curve

Here we show results of anytime performance comparisons between our approach and CSA. The reported results in the tables are computed using the same metric as described in Section C but we compare the AUC values instead of the final performance values. Table numbers here correspond to table numbers in the main paper. Table D1 shows the influence of different sampling rates on the performance, in terms of AUC, of our approach. This table confirms that also in terms of anytime performance a low sampling rate of 0.3 seems to perform best whereas smaller or larger sampling rates are detrimental.

Transfer to Longer Trajectories Table D2 shows the performance of our method when transferring to higher dimensions as well as to longer optimization trajectories. When transferring to longer optimization trajectories, our policy was trained for 500 function evaluations, i.e. 50 generations. We can see from Table D2a that our method is capable of outperforming the baseline CSA in terms of anytime performance. When we compare it to Table D2a of the main paper however we can see that the final performance gets worse, the further the optimization trajectory length is from the training setting. Plots in the following sections of this appendix show that, especially in the early stages of optimization, our learned policy very much outperforms the baseline. This early lead in optimization performance gives our method a much better AUC than the baseline.

Transfer to Higher Dimensions Table D2b shows the ability to transfer to higher dimensions, having trained the agent on functions of lower dimensions (i.e. 5D - 30D). We can see that up to 55D our learned policies AUC seems to be stay nearly the same. However for 60D the anytime performance becomes worse.

	Trajectory Length						
	50	100	150	200	250	500	1000
BentCigar	0.00	**1.00**	0.00	**1.00**	**1.00**	0.00	0.00
Discus	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Ellipsoid	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Katsuura	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rastrigin	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rosenbrock	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Schaffers	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Schwefel	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Sphere	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Weierstrass	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Average	0.90	**1.00**	0.90	**1.00**	**1.00**	0.90	0.90

(a) Different Trajectory Lengths

	Dimensions					
	35	40	45	50	55	60
BentCigar	0.00	0.00	0.00	0.00	**0.28**	0.00
Discus	**1.00**	0.00	**1.00**	**1.00**	0.00	0.00
Ellipsoid	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Katsuura	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.00
Rastrigin	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rosenbrock	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Schaffers	0.00	0.00	0.00	0.00	**1.00**	0.00
Schwefel	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Sphere	0.00	0.00	0.00	0.00	**1.00**	0.00
Weierstrass	0.00	**1.00**	0.00	**1.00**	**1.00**	**1.00**
Average	0.70	0.70	0.60	0.70	0.83	0.50

(b) Different # Dimensions

Table D2: Probability of our method to outperform the baseline (a) for varying trajectory lengths, when having only trained with trajectories of length 50, and (b) for different dimensions when training them on functions of dimension $5-30$ and applying the learned policies to functions of dimensionality > 30.

This stands in contrast to our analysis in the main paper, using only the final performance, where with increasing dimensionality we could observe better final values.

E Performance Comparison across Training Iterations

In this section we provide additional plots for Section 5.2 of the main paper. We visually compare the performance of our proposed method to the baseline at different iterations of the training process. Figures 1a to 1j depict the performance of our method after 1, 5, 10 and 15 training iterations as well as the baseline performance when training only on the respective functions.

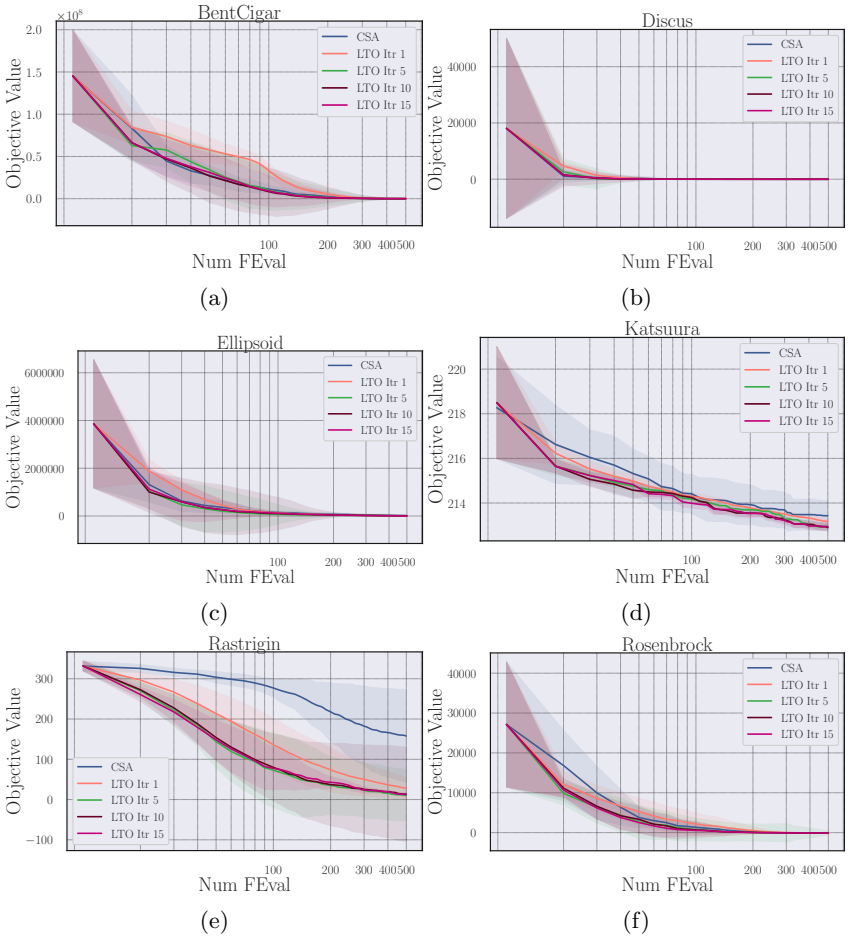


Fig. 1: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our methods incumbent policy after 1, 5, 10 and 15 training iterations of GPS on 10 BBOB functions.

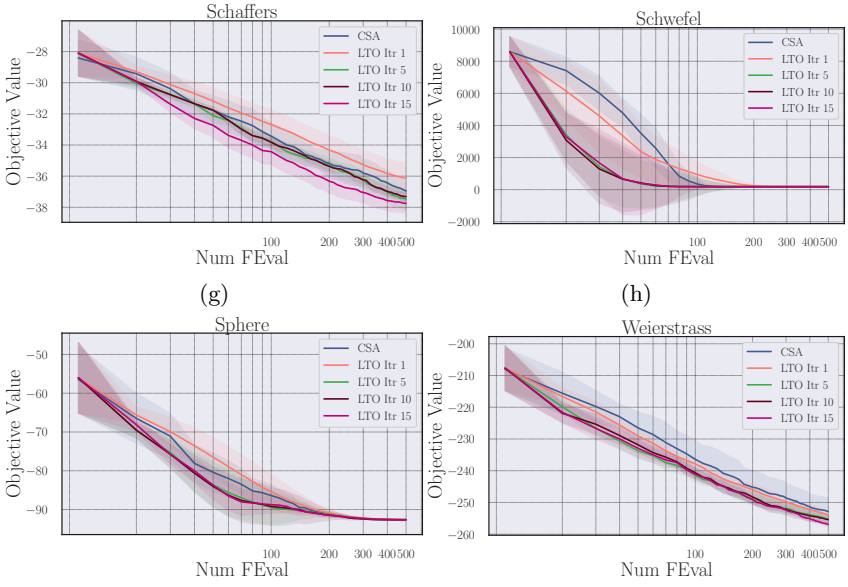


Fig. 1: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our methods incumbent policy after 1, 5, 10 and 15 training iterations of GPS on 10 BBOB functions.

F Sampling Rate

In this section we provide additional plots for Section 5.2 / Table 1 of the main paper. We visually compare the performance of our proposed method using a sampling rate of 0.3, vanilla GPS (i.e. sampling rate of 0) and the baseline (i.e. sampling rate of 1.0). Figure 2 shows the different optimization trajectories of CMA using the different step-size policies. Figure 3 depicts the corresponding step-size policies.

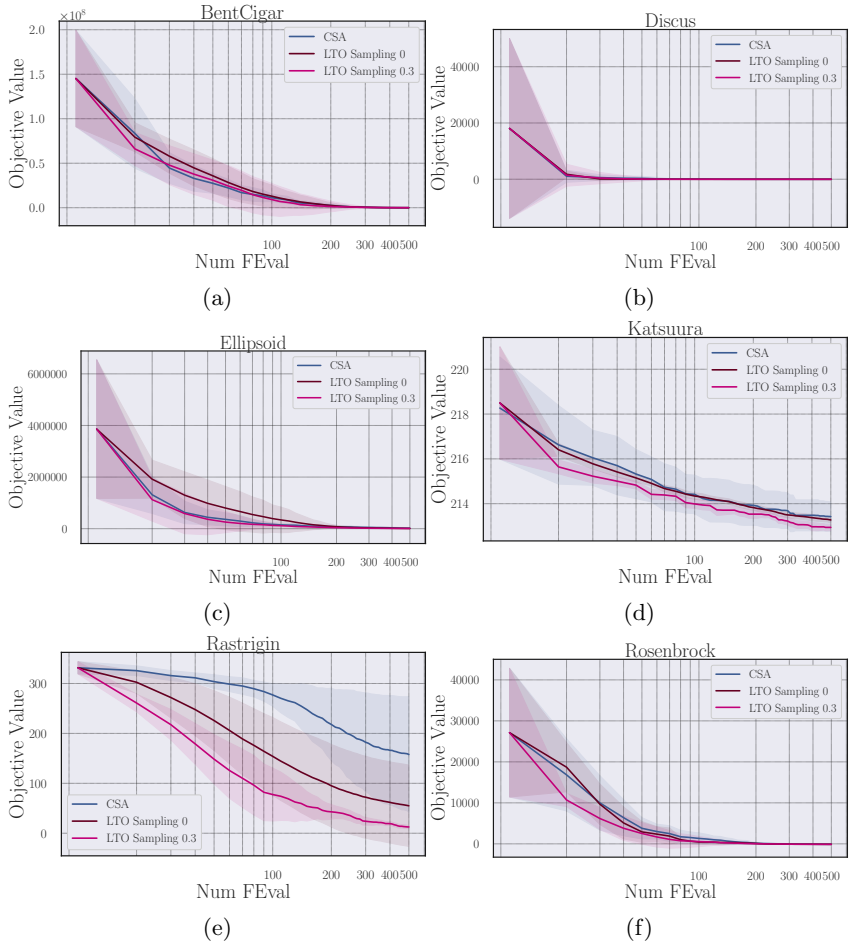


Fig. 2: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our method with a sampling rate of 0 and 0.3 on 10 BBOB functions.

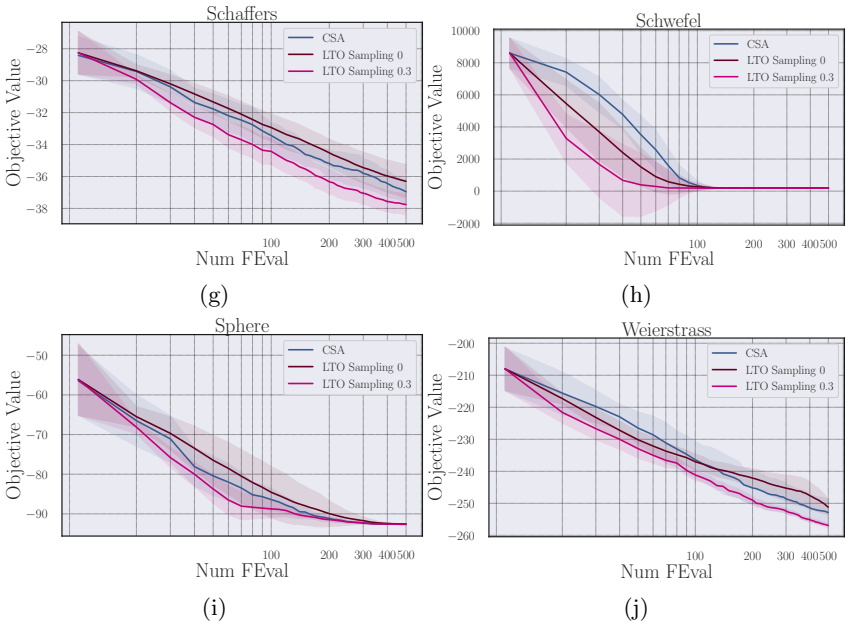


Fig. 2: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our method with a sampling rate of 0 and 0.3 on 10 BBOB functions.

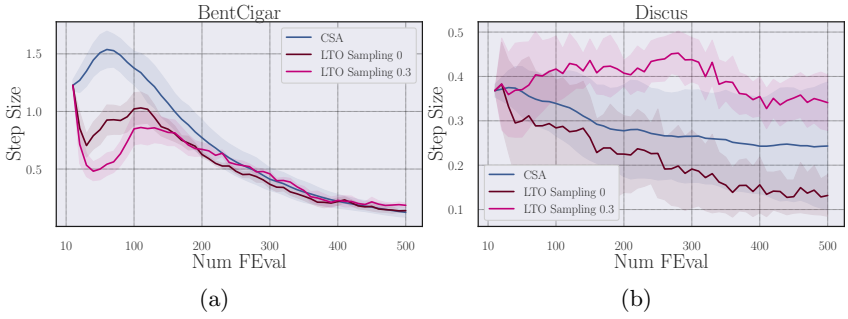


Fig. 3: Step-size adaptation comparison of CSA to that of our learned policies with a sampling rate of 0 and 0.3 on 10 BBOB functions.

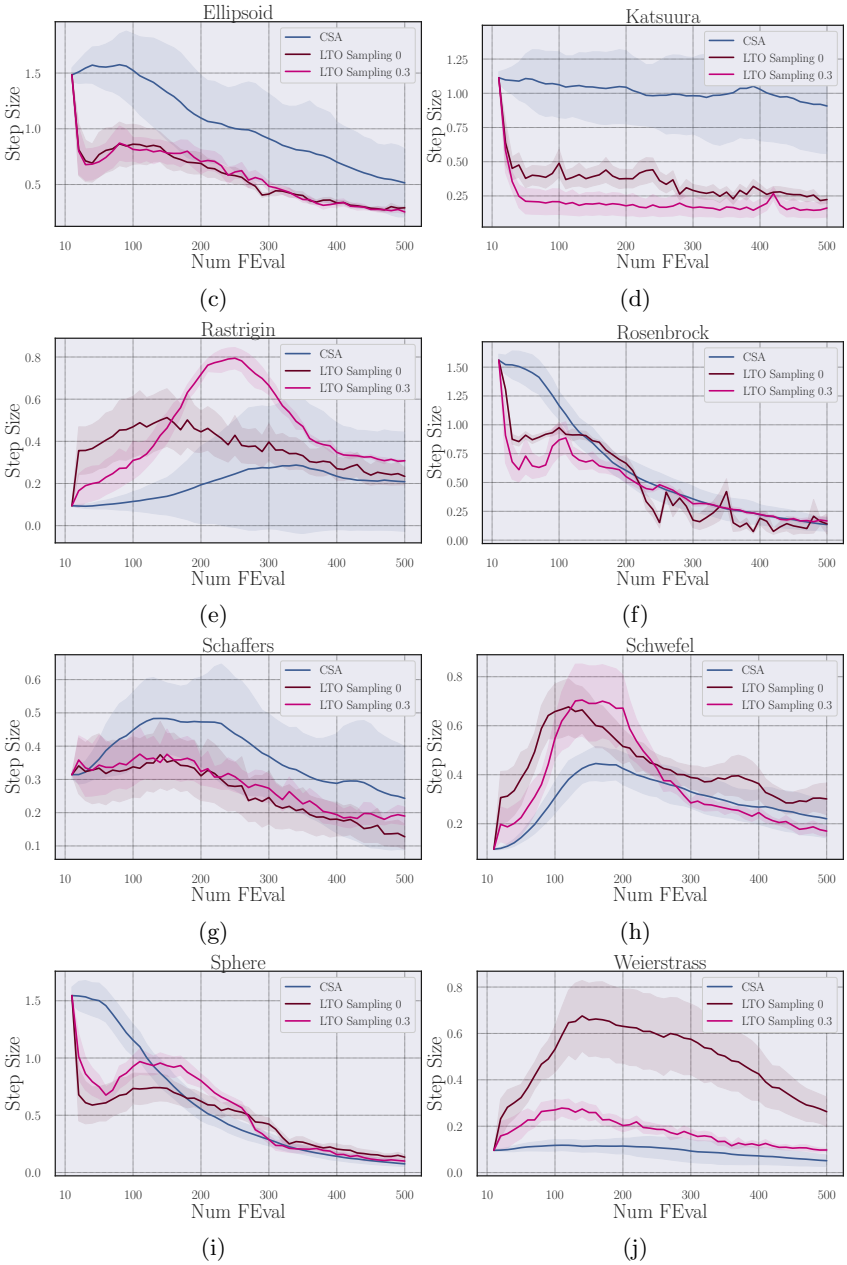
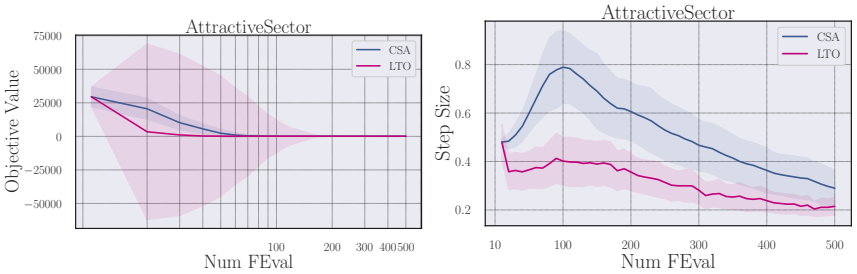


Fig. 3: Step-size adaptation comparison of CSA to that of our learned policies with a sampling rate of 0 and 0.3 on 10 BBOB functions.

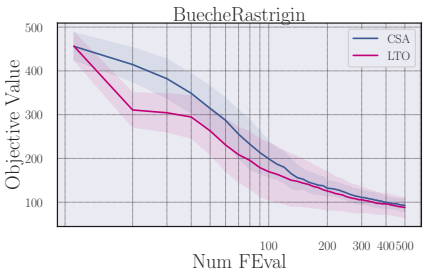
G Transfer to Unseen Test Functions

In this section we provide additional plots for Section 5.4 of the main paper. In the following plots we show the resulting optimization trajectory (left) when using the baseline and learned policy (right) on the corresponding function. We can see that, especially in the beginning, our learned policy learns to use smaller step-size values than CSA.

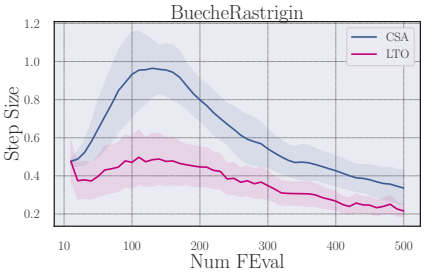


(a) Optimization Trajectory

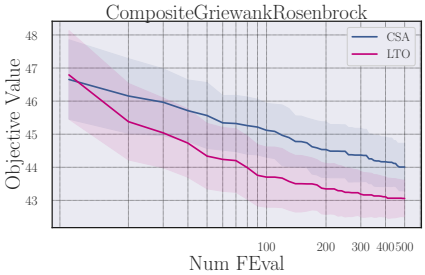
(b) Step-Size Policy



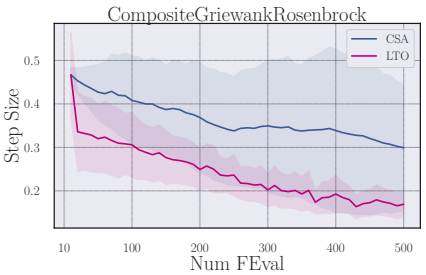
(c) Optimization Trajectory



(d) Step-Size Policy

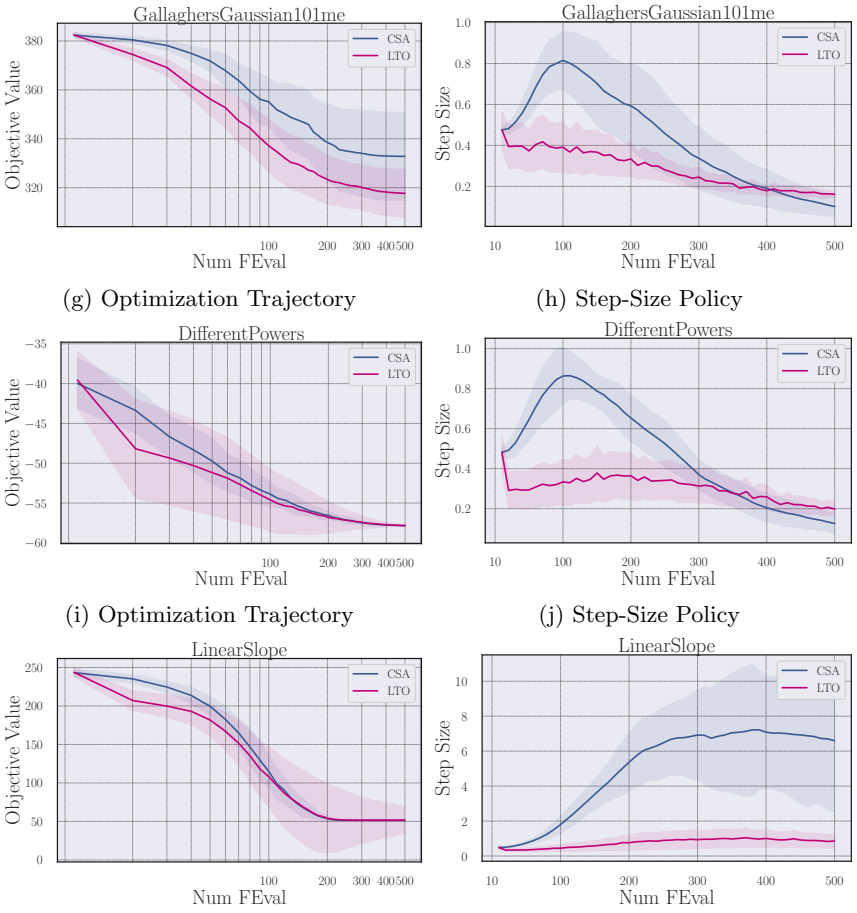


(e) Optimization Trajectory



(f) Step-Size Policy

Fig. 4: Optimization trajectories/Step-Size policies of CMA-ES using CSA (blue) and our learned policy (magenta) on 12 unseen test functions.

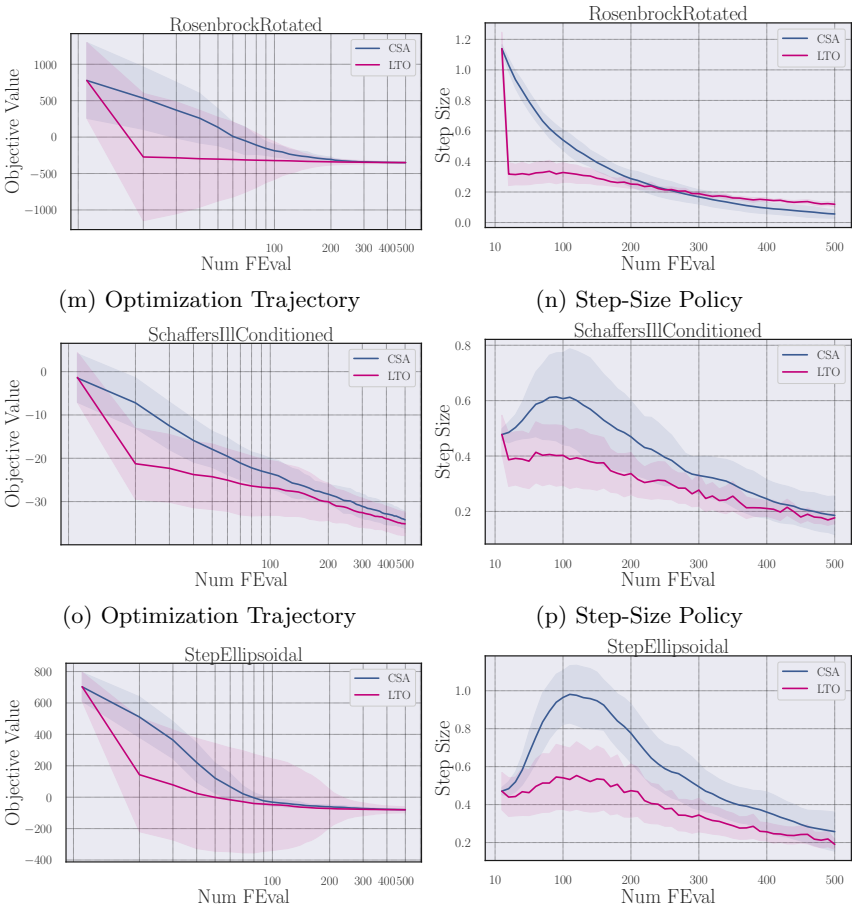


(g) Optimization Trajectory (h) Step-Size Policy

(i) Optimization Trajectory (j) Step-Size Policy

(k) Optimization Trajectory (l) Step-Size Policy

Fig. 4: Optimization trajectories/Step-Size policies of CMA-ES using CSA (blue) and our learned policy (magenta) on 12 unseen test functions.

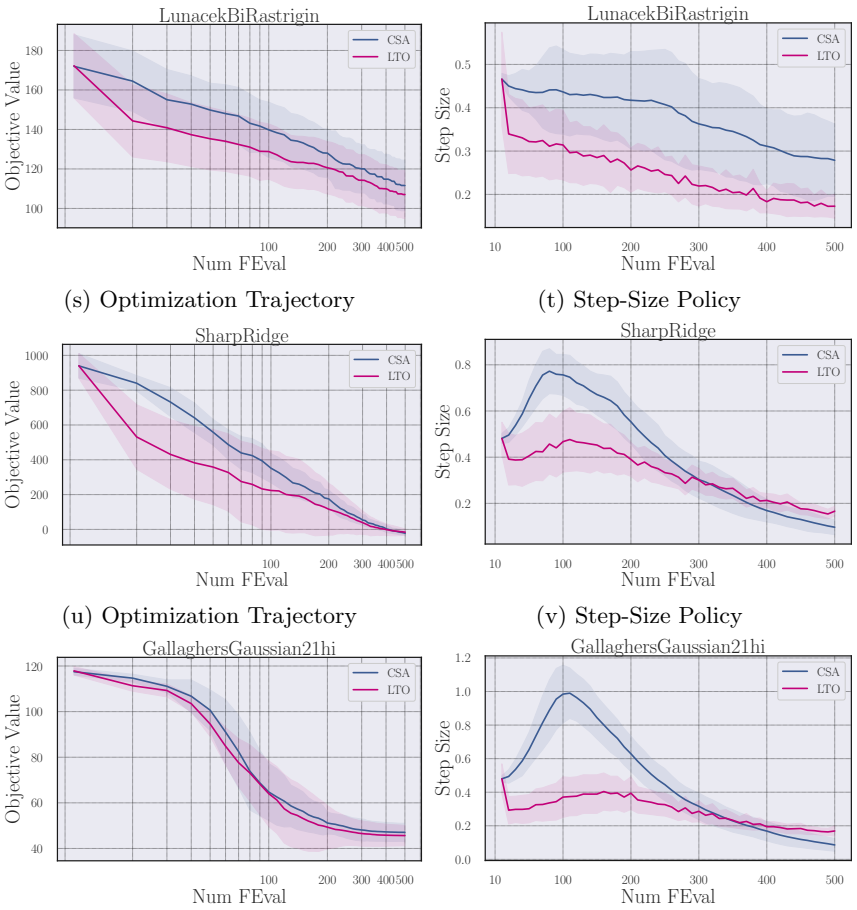


(m) Optimization Trajectory (n) Step-Size Policy

(o) Optimization Trajectory (p) Step-Size Policy

(q) Optimization Trajectory (r) Step-Size Policy

Fig. 4: Optimization trajectories/Step-Size policies of CMA-ES using CSA (blue) and our learned policy (magenta) on 12 unseen test functions.



(s) Optimization Trajectory (t) Step-Size Policy

(u) Optimization Trajectory (v) Step-Size Policy

(w) Optimization Trajectory (x) Step-Size Policy

Fig. 4: Optimization trajectories/Step-Size policies of CMA-ES using CSA (blue) and our learned policy (magenta) on 12 unseen test functions.

Appendix D

Appendix for Learning Heuristic Selection with Dynamic Algorithm Configuration

A Theoretical Results

Theorem 4. *For each algorithm selection policy $\tilde{\pi}_{as}$ there exists a family of planning instances i_n, a collection of heuristics H and a dynamic control policy $\tilde{\pi}_{dac}$, so that greedy best-first search with H and $\tilde{\pi}_{as}$ expands exponentially more states in $|i_n|$ than greedy best-first search with H and $\tilde{\pi}_{dac}$ until a plan π is found.*

Proof. Let $\tilde{\pi}_{\text{as}}$ be an algorithm selection policy. We consider the family of planning tasks i'_n, which is similar to the family of planning tasks i_n (Figure 2 in the main paper), with one modification: the goal state s_2 is not directly reachable from s_1, but via an additional state s'. In other words, we insert the state s' between s_1 and s_2. Furthermore, we again consider a collection of two heuristics $H = \{h_0, h_1\}$ with the heuristic estimates shown in Figure 2 (main paper) and $h_0(s') = 2$ and $h_1(s') = 10$. The idea is that both heuristics alone lead to the expansion of exponentially many states, whereas a dynamic switch of the heuristic only leads to constantly many expansions.

Policy $\tilde{\pi}_{\text{as}}$ selects exactly one heuristic, h_0 or h_1, for each planning task. If h_0 is selected, with the same argument used in the proof of Theorem 3 (main paper), exponentially many states in $|i'_n|$ are expanded. If h_1 is selected, in time step 2, states s_3 and s' are contained in both open lists. According to h_1, state s_3 is more promising than s', which leads again to an expansion of exponentially many states in $|i'_n|$.

In comparison, for $\tilde{\pi}_{\text{dac}}$ we pick again the policy that always selects the heuristic with minimum average heuristic value of all states in the corresponding open list, i.e. $\arg\min_{h \in H} \mu_h$. Policy $\tilde{\pi}_{\text{dac}}$ selects first h_0, followed by h_1 and again h_0, resulting in the generation of the goal state after three state extensions. □

B White-Box Experiments

We conducted preliminary experiments on a newly created ARTIFICIAL domain with two artificial heuristics. This domain is designed so that in each step, only one of two heuristics is informative. In other words, similar to the constructed example in the proof of Theorem 4, at each time step, only one heuristic leads to the expansion of a state which is on the shortest path to a goal state. In order to obtain a good control policy that leads to few state expansions, it is necessary to derive a *dynamic* control policy from the state features. We generated 30 training instances on which we performed a small grid search over the following parameters *#layers* $\in \{2, 5\}$, *hidden units* $\in \{50, 75, 150, 200\}$ and *epsilon decay* $\in \{2.5 \times 10^5, 5 \times 10^5\}$. We determined that a 2-layer network with 75 hidden units and a linear decay for ϵ over 5×10^5 steps from 1 to 0.1 worked best [5].

Interestingly, it was possible to learn policies with a performance close to the optimal policy, see Figure B1. Both individual heuristics perform poorly (even when using an oracle selector). Randomly deciding which heuristic to play

[5]Note that the hyperparameters for experiments on the IPC domains have not been further tuned.

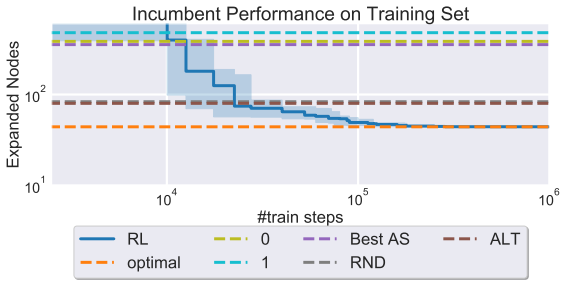


Figure B1: Performance of the best learned policy during training (RL), compared to the the performance of the individual heuristics (0 & 1), the oracle selector (BEST AS), an alternating schedule (ALT), a random policy (RND) and the optimal policy. Dashed lines indicate the performance of our baselines, the solid line the mean performance and the shaded area the standard deviation of our approach.

performs nearly as good as the alternating strategy that alternates between the heuristics at each step. In the beginning the learned policy needs some time to figure out in which states a heuristic might be preferable. However, it quickly learns to choose the correct heuristic, outperforming all other methods and nearly recovering the optimal policy.

C Domain Dependence

We limit ourselves to learning domain-dependent policies as it was uncertain if learned policies would be able to transfer from training with a cutoff to the test set with potentially much longer trajectories. In particular, we used a conservative cutoff of 7500 node expansions for training. When evaluating our fully trained policies, we did not use this node expansion cutoff, but ran FAST DOWNWARD for 300 seconds (allowing for potentially many more than 7500 node expansions). To gain insights into how many of the training and testing problems could be solved by our learned policies without use of the conservative cutoff we evaluated all our learned policies with the cutoff of 300 seconds. On the *training* set roughly 31% of all training instances that can be solved within 300 seconds by our policies require more than 7500 node expansions. On the test set it increases slightly to 33% of the solved instances require more than 7500 expansions. It is worth noting that on the test sets for BARMAN and VISITALL this percentage is even 49% and 66% respectively. As this generalization to unseen instances with potentially much longer trajectories was a first significant hurdle to overcome we limited ourselves to domain-dependent policies. In future work we will remove this limitation.

D Additional Experiments

We conducted additional experiments with 5 heuristics instead of 4, also including the $h_{\text{lm-count}}$ heuristic. Overall, the results are similar to those presented in the paper, showing that the learned DAC policy performs best overall. Note that our computer cluster has been updated to Ubuntu 20.04 LTS 64 bit in the meantime. Therefore, we ran all the configurations again, including the static ones, which led to slightly

Algorithm	CONTROL POLICY			SINGLE HEURISTIC					BEST AS (ORACLE)		
Domain (# Inst.)	**RL**	RND	ALT	h_{ff}	h_{cg}	h_{cea}	h_{add}	$h_{\text{lm-count}}$	**RL**	ALT	SINGLE h
BARMAN (100)	**85.6**	84.2	84.1	64.0	16.0	18.0	18.0	77.0	**91.2**	85.0	85.0
BLOCKSWORLD (100)	**95.0**	89.8	90.8	75.0	60.0	92.0	92.0	75.0	**99.0**	93.0	97.0
CHILDSNACK (100)	84.4	82.0	83.4	75.0	**86.0**	**86.0**	**86.0**	68.0	**89.0**	85.0	86.0
ROVERS (100)	**100.0**	**100.0**	**100.0**	83.0	72.0	67.0	67.0	**100.0**	**100.0**	**100.0**	**100.0**
SOKOBAN (100)	87.0	87.1	87.0	88.0	**90.0**	60.0	88.0	85.0	87.0	87.0	**92.0**
VISITALL (100)	**100.0**	**100.0**	**100.0**	38.0	60.0	59.0	60.0	**100.00**	**100.0**	**100.0**	**100.0**
SUM (600)	**552.0**	543.1	545.3	423.0	384.0	382.0	411.0	505.0	**566.2**	550.0	560.0

Table D3: Average coverage of different policies for the selection of a heuristic in each expansion step when evaluating the strategies on the prior unseen *test* set. The first three columns are control policies, the next *five* are individual heuristic searches, while the last three represent the best algorithm selection of the corresponding strategies, i.e., oracle selector for each instance.

different results here than in the main experiments.

E Additional Analysis

Tables E5 and E4 shows the average heuristic of a run. Every section of the table refers to one quarter of a full policy trajectory in which a heuristic is selected at every step. We can observe that the learned policies tend to favor one of the heuristics over all others. For example in BARMAN, on average the learned policies select the h_{ff} heuristic more than 50% of a successful run, whereas it selects h_{cg} roughly for one third of a successful run. Further, we can observe that in most domains the learned policies tend to focus on two out of the four available heuristics. Only in CHILDSNACK and SOKOBAN do we observe that a single heuristic is chosen throughout the entire run.

By splitting up the observed trajectories in quarters and analyzing these quarters individually, we can see that the learned heuristics tend to slightly increase the usage of the dominant heuristic over time. In BLOCKSWORLD for example, in the first quarter h_{add} is selected roughly 56% on average and increases to 60% in the second quarter, before leveling out at 61% in the third and fourth quarters. Only on ROVERS we observe an inverse of this trend where h_{ff} is selected 71% in the first quarter before dropping to 68% and 67% in the third and fourth quarter respectively. Sometimes this increase in usage of the more dominant heuristic is reflected with an appropriate reduction in usage of the second dominant heuristic, (see e.g. BARMAN). In other cases however, both dominant heuristics increase in usage while reducing the usage of the mostly unused heuristics (see e.g. BLOCKSWORLD).

Table E6 shows the average switching frequency observed on the test data of the individual IPC domains. For this analysis, we recorded the length of heuristic usage before switching to another heuristic. For ease of analysis, we differentiate between the four different frequency classes:

- Immediate $\Rightarrow$ switched heuristics already after one step;
- High $\Rightarrow$ switched after 2 to 100 steps;
- Medium $\Rightarrow$ switched after 101 to 1000 steps;
- Low $\Rightarrow$ switched after more than 1000 steps.

Here we report the average, maximal and minimal switching frequencies observed on the solved instances of the individual domains. This allows us to gain insights into how often switching was necessary. On BARMAN we can for example observe that on average, 28% of the performed steps in this domain were immediate switches between heuristics. Further, we can also see that on every problem instance, some of the performed steps were immediate switches between heuristics (see column "Min" under "Immediate"). This is reflected accordingly in the maximum values of any of the frequency classes as none is ever higher than 60%. For all other domains however, we can see that a successful did commit to longer trajectories for at least one test problem instance.

For some domains, such as SOKOBAN or CHILDSNACK we can see that at least one instance in the domain was solved by a policy, by consistently playing only one of the possible heuristics. This is consistent with the reported coverage and heuristic usage values in Tables 1 of the main paper, as well Table E5 of this section. Lastly, we can observe that the lowest frequency class is observed least frequently. This indicates that the our agents did mostly commit to such long repeated actions if they learned that a problem instance is solved best by not switching.

	h_{ff}			h_{cg}			h_{cea}			h_{add}		
Domain (# Solved Inst.)	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min
BARMAN (84.4)	**0.60**	0.87	0.21	0.38	0.77	0.13	0.02	0.05	0.00	0.00	0.01	0.00
BLOCKSWORLD (92.9)	0.02	0.10	0.00	0.00	0.02	0.00	0.38	0.50	0.00	**0.60**	1.00	0.47
CHILDSNACK (88.)	**0.95**	1.00	0.62	0.02	0.20	0.00	0.01	0.04	0.00	0.02	0.16	0.00
ROVERS (95.2)	**0.68**	0.92	0.56	0.05	0.15	0.00	0.18	0.26	0.04	0.09	0.13	0.04
SOKOBAN (87.7)	**0.98**	1.00	0.80	0.00	0.20	0.00	0.02	0.03	0.00	0.00	0.02	0.00
VISITALL (56.9)	0.02	0.20	0.00	0.18	0.33	0.00	**0.43**	1.00	0.04	0.37	0.60	0.00

Table E4: Overall heuristic selection by RL per expansion step, averaged over solved instances. We highlight in bold the most frequently selected heuristic for expansion per domain on average.

	Q1				Q2				Q3				Q4			
Domain (# Solved Inst.)	h_{ff}	h_{cg}	h_{cea}	h_{add}	h_{ff}	h_{cg}	h_{cea}	h_{add}	h_{ff}	h_{cg}	h_{cea}	h_{add}	h_{ff}	h_{cg}	h_{cea}	h_{add}
BARMAN (84.4)	**0.56**	0.41	0.03	0.00	**0.61**	0.37	0.02	0.00	**0.62**	0.36	0.02	0.00	**0.62**	0.36	0.02	0.00
BLOCKSWORLD (92.9)	0.07	0.01	0.36	**0.56**	0.01	0.00	0.38	**0.60**	0.01	0.00	0.39	**0.61**	0.00	0.00	0.39	**0.61**
CHILDSNACK (88.)	**0.91**	0.03	0.02	0.04	**0.95**	0.02	0.00	0.03	**0.97**	0.01	0.00	0.02	**0.97**	0.01	0.01	0.01
ROVERS (95.2)	**0.71**	0.03	0.16	0.10	**0.68**	0.05	0.18	0.09	**0.67**	0.05	0.19	0.09	**0.68**	0.05	0.18	0.09
SOKOBAN (87.7)	**0.97**	0.00	0.02	0.01	**0.98**	0.00	0.01	0.01	**0.98**	0.00	0.01	0.01	**0.99**	0.00	0.01	0.00
VISITALL (56.9)	0.05	0.18	**0.43**	0.34	0.01	0.19	**0.42**	0.38	0.01	0.18	**0.43**	0.37	0.02	0.18	**0.44**	0.36

Table E5: Average heuristic usage over all solved test problem instances split into quarters of the learned policies. Bold-faced entries are the most frequently used heuristic per quarter.

	Immediate			High Freq.			Medium Freq.			Low Freq.		
Domain (# Solved Inst.)	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min
BARMAN (84.4)	0.28	0.51	0.11	**0.57**	0.76	0.26	0.10	0.23	0.09	0.05	0.49	0.00
BLOCKSWORLD (92.9)	0.14	0.56	0.00	**0.40**	0.95	0.04	0.28	0.67	0.00	0.18	0.85	0.00
CHILDSNACK (88.)	0.08	0.31	0.00	0.33	0.95	0.00	**0.41**	0.99	0.00	0.18	0.99	0.00
ROVERS (95.2)	0.28	0.54	0.00	**0.60**	0.86	0.01	0.06	0.55	0.00	0.06	0.98	0.00
SOKOBAN (87.7)	0.04	0.24	0.00	**0.50**	0.98	0.01	0.33	0.98	0.00	0.13	0.80	0.00
VISITALL (56.9)	0.07	0.30	0.01	0.24	0.85	0.03	0.28	0.66	0.00	**0.41**	0.89	0.00

Table E6: Switching frequency of the learned policies on the test dataset. Bold-faced entries give the, on average, most used switching frequency per domain.

Appendix E

Appendix for DACBench: A Benchmark Library for Dynamic Algorithm Configuration

C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *Proc. of ICML*, pages 7105–7114, 2019.

A Structure & Implementation

Section 4.2 of the main paper discussed necessary design decisions of a DAC benchmark. Here we describe in greater detail how we chose to implement these ideas in practice.

Structure A crucial first decision that shaped the project is the structure of DACBench. It is designed to enable researchers to modify benchmarks more easily than currently possible, but also to replicate diverse experiment settings quickly. To accomplish this, we initialize the benchmark environments with an experiment configuration that can be stored and shared easily. Researchers can then load this configuration and reproduce the experiment (see also Section C). The benchmark configuration focuses on the specification options for mentioned in Section 4.1 of the main paper.

Algorithm Implementation We provide or specify the implementation as well the definition of what constitutes a *step* in this implementation in order to keep execution consistent. As researchers may not focus on dynamically optimizing all hyperparameters in parallel, the hyperparameters of the target algorithm that are not currently controlled can be modified prior and are kept fixed throughout a run. This can also include utility parameters concerning the DAC interface, parallelization or using different execution modes of the target algorithm.

Action Space Here the user can specify the search space for the hyperparameters being controlled. It is important that a step of the target-algorithm knows how to parse the parameter values. Otherwise, the order of parameters might get mixed and can lead to faulty behaviour.

State & Reward Researchers can use the default methods for computing state or reward, but they can also provide custom ones that better fit their needs. DACBench can record and reconstruct experiments with custom functions as long as their implementation is shared with the experiment description.

Instance Sets Lastly, we provide instance sets for training and testing, either directly from published versions of the benchmarks or sampled from distributions that are as similar as possible. The training and tests sets are drawn from the same distribution and contain 100 instances each. We believe using fixed instance sets will make comparisons between methods more consistent overall. Nevertheless, of course different instance sets as well instances sampled on the fly can be used to better explore how DAC methods behave in more general cases or specific regions of the instance space. While we strive to record experiment specifications as succinct as possible, the instances themselves will have to be shared by the researchers working with them. Additionally users can modify the experimental setup including the seed, utilities for logging results and other relevant information like additional functionality like added noise. These are often important for reproducibility and should be shared with other researchers along with the benchmark specification itself.

Interface The interface we use for conducting these experiments is heavily based on the OpenAI gym Brockman *et al.* [2016] interface for its simplicity. Users only need to initialize the environment and the call two methods, reset and step, to interact with it. Furthermore, it provides an easy way to specify action and state spaces. As this the most commonly used RL interface, it will be very easy for RL experts to work on DAC, making the field as a whole more approachable. To ensure proper initialization using the configuration, environments in DACBench are created by benchmark classes, each holding a default configuration that can be expanded upon and saved. Additionally, we provide an evaluation method for the whole benchmark suite, to make comparisons to baselines and other methods simple and consistent. We provide data of standard baselines like random policies to standardize these comparisons further. As a lot of the benchmark specification is done through the configuration, smaller changes like the use of a different state function can be done by domain experts without modifying the environment code. We believe this will benefit the field in the long run as experts can improve benchmarks with little effort and without working through all of the existing code base. This will lead to better benchmarks and thriving community around DAC.

Additional functionality We provide functionality to add randomness to the benchmarks in the form of sampled instances from custom distributions as well as noise on the reward signal. We do not include these in the default benchmarks to keep execution consistent but they provide important insight in algorithm behaviour nonetheless. Moreover, we include smaller benchmark suites that provide both research direction and progress measure in a very efficient format. Lastly, DACBench also includes a flexible logging system that allows the user to track performance data as well as action trajectories, state features and execution times, all sorted by environment step and instance used. To make analysis quick and convenient, we include methods for reloading data as well as visualization tools.

B The Benchmarks in detail

Sigmoid The Sigmoid benchmark [Biedenkapp *et al.*, 2020] is parameterized by the number of functions to approximate in parallel in the number of actions in each of these dimensions (by default two dimensions with 10 and 5 actions respectively). An instance provides shift and slope for the sigmoid function in each of these dimensions such that the function value is:

$$sigmoid(t) = \frac{1}{1 + e^{-slope*(t-shift)}}$$

The reward in each dimension is then the distance from the chosen action to the function value. The rewards from each dimension are then multiplied to form the step reward. The state is made up of the current step, previous state (including all slopes and shifts for the current instance), action played, reward and next state.

Luby This benchmark has the agent approximate the Luby sequence of a given length. (Following Biedenkapp *et al.*

[2020] the length is pre-set to 64, resulting in 6 distinct actions). In the default setting, this is challenging enough, but instances for Luby can be used to modify the start value of the sequence or add an accumulating error signal. The reward is 0 if the correct action is played, -1 otherwise. The observations returned consist of the current timestep, previous state (including a history of actions for the last 5 steps), action played, reward, next state, next goal and time cutoff.

FastDownward In FastDownward, the agent selects a search heuristic for the next planning interval. The default action space consist of two heuristics, although we also provide a more complete version with four heuristics [Speck *et al.*, 2020]. The two heuristic version is an easier variation with artificial instances that provides an easier optimization variation and it has been shown that an RL agent is capable of recovering the optimal policy on these [Speck *et al.*, 2020]. The reward is -1 per step, measuring the total number of steps. The agent observes average, minimum and maximum values as well as variance and open list entries for each heuristic. We provide several target domains as instance sets with the default being an artificially generated one.

CMA-ES Following Shala *et al.* [2020], an agent's task here is to adjust the step size for CMA-ES. Therefore the actions space covers possible step sizes between 0 and 10. The reward is the best individual's objective value, with the observations consisting of the current step size, cumulative path length and population size as well as past objective values, change in past objective values and past step sizes (for the last 40 steps each). The instance set consists of 10 different function classes of the BBOB benchmark.

ModEA Instead of controlling a single hyperparameter, in ModEA the algorithm structure consisting of 11 different components is adjusted [Vermetten *et al.*, 2019]. There are two choices for each of the first $nine$ of these and three for the other two. That results in $4,608$ possible actions in total. The reward is, as in CMA-ES, the best individual's fitness. The state contains the generation size, current step size, remaining budget, function ID and instance ID. As the large action space makes this benchmark hard, we used the same instance set as for CMA-ES with only 10 different function classes.

SGD-DL Here the agent controls a small neural network's learning rate Daniel *et al.* [2016]. Instances consist of a dataset, seed and network. In the default setting, we consider a single dataset (MNIST), 100 seeds each for training and test, affecting weight initialization and the mini-batch ordering, and a single fully-connected feedforward network architecture having two hidden layers with 16 units each. Permitted actions lie between 0 and 10 with the learning rate of the optimizer (Adam) being set to $10^{-action}$. The observations includes discounted average and uncertainty each of the predictive change variance and loss variance, the current learning rate, the training and validation loss.

C Modifiers of Benchmarks

DACBench not only allows to use existing benchmarks, but also enables easy modification through the use of Benchmark Configurations themselves (see Figure C1).

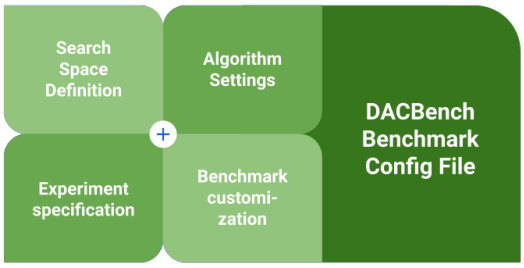


Figure C1: Modification possibilities.

Modifications to the search space definition include changes to the state and action spaces. Such changes can immediately increase or lower the difficulty of learning successful policies, e.g. through inclusion/exclusion of irrelevant action choices or use of additional (un-)informative state features. Changing the algorithm settings include changes to hyperparameters that are not dynamically changed or different resource allocation. Further benchmark customization could allow control of multiple parameters (if not already enabled) or use of different problem instances. All included benchmarks come with default configurations that allow for reproduction of experiments from published versions of the benchmarks.

D Experiment Hardware

All experiments with were conducted on a slurm CPU cluster (see Table D1). The upper memory limit for these experiments was 800MB per run, although not all benchmarks require this much. As the DACBench runner was used for all experiments, the provided Slurm scripts can reproduce the results. Additionally, we provide them with the code.

Machine no.	CPU model	cores	RAM
1	Xeon E5-2670	16	188 GB
2	Xeon E5-2680 v3	24	251
3-6	Xeon E5-2690 v2	20	125 GB
7-10	Xeon Gold 5120	28	187

Table D1: CPU cluster used for experiments.

E On Quantifying the Challenge Dimensions

We quantify all of our dimensions of difficulty for better comparability. This section details our criteria.

State & Action Spaces First, state and action space size are deciding factors in MDPs. If state or action spaces are larger, learning will take longer and the probability of finding a local optimum instead of the best solution could increase for many methods. Therefore a small set of possible states and actions makes a benchmark easier to solve regardless of how complex the underlying function is. To make comparison between the other aspects as well as discrete and continuous action easy, we divide the spaces into three categories. Category one contains small discrete action spaces, we define this as below

100 actions. Large discrete spaces with up to 1000 actions fall into category two. For state spaces, this means spaces of up to 100 dimensions. Larger spaces fall into category three. Continuous action spaces and action spaces with more than 1000 possible actions fall into the same category. This is of course only a very rough categorization, but it should provide an overview of how our benchmarks differ.

Reward Function In building DAC benchmarks, deciding on a reward signal is as important as it can be difficult. A good reward signal would attribute every action a reward proportional to its quality. This is hard to accomplish and sometimes we have to default to a very sparse reward signal, requiring the agent to learn to interpret the reward. A more informative reward, what we call better reward quality, is therefore a desirable quality from a learning perspective. For this category, we define a scale from 1 and 5: 5 means no meaningful reward at all, 4 is a combined reward only at the end with no information (reward of 0 in each step) during the run. A score of 3 is similar, a meaningful signal only at the end but with step rewards that indicate if taking steps is desired or not (e.g. now giving $+1$ or -1 per step). A reward of quality 2 provides the accumulated quality of the policy so far at each step, but not how the last action contributed specifically. Lastly, if the reward indicates directly how good of a choice the last action specifically was, it would be of quality level 1. Just like the action and state space size, we can judge this benchmark feature without any empirical evaluations.

Noise As DAC means working with algorithms that may not have exact same execution times and patterns across different runs and hardware setups, a DAC agent should be able to learn and perform in noisy settings. Therefore we consider reward noise, which makes finding the target policy harder, an important challenge in DAC. To measure it, we need to run the benchmarks. We chose random policies and compute the standard deviation normalized by the mean between 10 evaluations of the baseline policies per seed and average over 10 seeds.

Policy heterogeneity Policy heterogeneity is another component of benchmark difficulty. If policies across benchmark instances stay relatively similar, they should be easier and faster to learn, as all instances provide the same or at least a similar signal for optimizing the policy. As we do not have access to the optimal policy for our benchmarks, we use the results of our static policy evaluation as an estimation of how well a single policy can cover the instance space. We compare the average standard deviation normalized by the mean static policies show across all instances.

Dynamicity Lastly, we examine how dynamic our benchmarks are, that is how many action changes we expect in a policy. If a benchmark is not very dynamic, policies that only update the hyperparameter value once or twice might be best while highly dynamic benchmarks require almost constant changes. Again we lack the optimal policies to get a definite answer to this question, but we approximate it using static and random policies. For each benchmark, we cover the given search space with a number of static policies and run random policies with repeating actions. Actions are repeated for a total of 1, 10, 100 or 1000 steps. As before, we evaluate each repeat number for 10 seeds with 10 runs each. The benchmarks are scored depending on the performance ranks of these random policies. If the policy with only 1 repetition performs best on average on an instance per seed, its score is increased by 3. 10 repetitions yield 2 points, 100 1 point and no points if the policy with 1000 repeats performs best. We then scale this score to $(0, 1)$, as we do with all others, for simplicity's sake.

F Additional experimental results

For space reasons, we did not include all comparisons of static and dynamic policies in the main paper. As it is still interesting to see how dynamic baselines and random policies compare to the best and worst static policies, we include the missing benchmarks here. Within these, we can easily identify two groups: on the Sigmoid and FastDownward benchmarks, dynamic policies obviously perform well. FastDownward in particular seems to favor heavily dynamic policies, as our previous analysis has shown already. For ModEA and CMA-ES, the picture is not quite so clear, with the random policy and even CSA for CMA-ES peforming somewhere in between the best and worst static policies. As we have seen results on these problem settings that suggest good dynamic policies perform far better, however, we can simply assume CMA-ES and ModEA to be harder benchmarks to beat static policies on. As they both have large action spaces as well as high policy heterogeneity and noise rating in addition to this fact, they present the upper end of difficulty in DACBench.

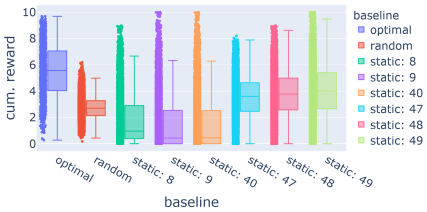


Figure F1: Static and dynamic policies on Sigmoid including top & bottom 3 static policies. The reward measures how close the chosen discrete value is to the actual funvtion value.

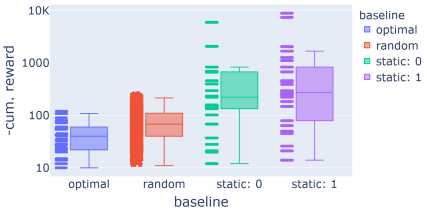


Figure F2: Static and dynamic policies on FastDownward. The reward is -1 per step until the run is finished.

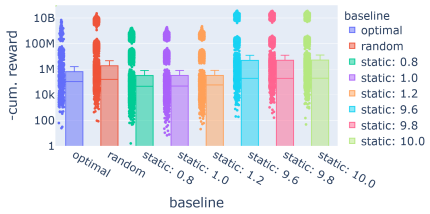


Figure F3: Static and dynamic policies on CMA-ES, including top & bottom 3 policies and CSA. The reward is the fitness of the best indivudual in the population.

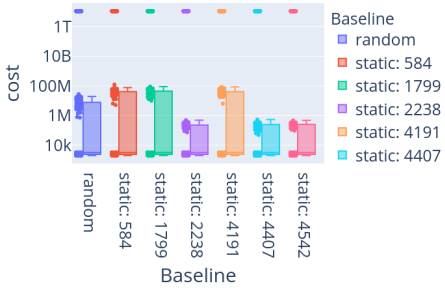


Figure F4: Static and dynamic policies on ModEA, including top & bottom 3 policies. The reward, as in CMA-ES, is the current best individual.

Appendix F

Appendix for TempoRL: Learning When to Act

A. Detailed Baseline Description

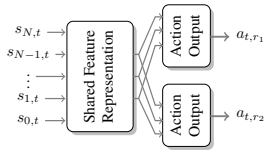


Figure A1. Schematic DAR Architecture with duplicate output heads to learn at two timescales r_1 and r_2.

Dynamic Action Repetition (DAR; Lakshminarayanan et al., 2017) is a framework for discrete-action space deep RL algorithms. For a discrete-action space $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$ DAR duplicates this space such that an agent can choose from $2 \times |\mathcal{A}|$ actions. Further, DAR introduces two hyperparameters r_1 and r_2, each of which are associated with one half of the new action space. These hyperparameters determine the number of time-steps an action will be played for, with both actions a_k and a_{2k} $(1 \leq k \leq |\mathcal{A}|)$ performing the same behaviour but a_k is repeated for r_1 time-steps and a_{2k} for r_2 time-steps. When training an agent, there are no modifications to the training procedure, other than an agent now having to select from a larger action space. Figure A1 schematically depicts a DAR DQN agents $\mathcal{Q}$-network architecture.

This gives an agent two levels of control to decide on how long to apply an action. A drawback of this framework is that the output heads are independent from each other and are not aware that certain action outputs have the same influence on the environment for $\min(r_1, r_2)$ time-steps. Further, both r_1 and r_2 have to be defined beforehand, requiring good prior knowledge about the potential levels of fine and coarse control in an environment.

Fine Grained Action Repetition (FiGAR; Sharma et al., 2017) is a framework for both discrete and continuous action spaces. Instead of learning a single policy that has to learn both which action to play and how long to follow it (as in DAR), FiGAR decouples the behaviour and repetition learning by using two separate policies $\pi_a : \mathcal{S} \to \mathcal{A}$ and $\pi_r : \mathcal{S} \to \{1, 2, \ldots, \text{max repetition}\}$. When training an agent, based on a state s, π_a decides which action to play and simultaneously π_r decides how long to repeat a selected action starting from s. At the time of selecting their respective actions, neither π_a nor π_r are aware of the other policies decision. Thus, the action and the respective repetition value are selected independently from each other.

To couple the learning of both policies Sharma et al. (2017) use a joint loss to update the network weights and further suggest to use weight-sharing of the input-layers of the two policy networks. Although this aligns the policies when performing a training step, at decision time the policies remain uninformed about each others behaviour. Counter to DAR, FiGAR allows for much more fine-grained control over the action repetition. However, FiGAR requires more modification of a base algorithm to allow for learning of control at different time-steps. With TEMPORL we propose a method that allows for the same fine-grained level of control while requiring no modifications to the base agent architecture.

Algorithm 1 TEMPORL $\mathcal{Q}$-learning

1: **Input:** environment env with states $\mathcal{S}$ and actions $\mathcal{A}$, skip-Actions $\mathcal{J}$,
behaviour and skip $\mathcal{Q}$-functions $\mathcal{Q}(\cdot,\cdot)$, $\mathcal{Q}(\cdot,\cdot|\cdot)$, training episodes E
2: Initialize $\mathcal{Q}(s,a)$, $\mathcal{Q}(s,j|a) \forall s \in \mathcal{S}, a \in \mathcal{A}, j \in \mathcal{J}$
3: **for** episode$\in \{1,\dots,E\}$ **do**
4: $s \leftarrow env.reset()$
5: **repeat**
6: $a \leftarrow \pi(s)$ # e.g. ϵ-greedy $\arg\max_{a' \in \mathcal{A}} \mathcal{Q}(s,a')$
7: $j \leftarrow \pi_j(s,a)$ # e.g. ϵ-greedy $\arg\max_{j' \in \mathcal{J}} \mathcal{Q}(s,j'|a)$
8: trajectory $\leftarrow [s]$ # Tracks the skip trajectory
9: **repeat**
10: $r, s' \leftarrow env.step(a)$
11: append s' to trajectory # Records the state transitions
12: $\mathcal{Q}(s,a) \leftarrow td\_update(\mathcal{Q}(s,a), r, s')$ # See Equation 5
13: $s \leftarrow s'$
14: **until** all skips $1,\dots,j$ performed or episode ends
15: $\mathcal{G} \leftarrow build\_connectedness\_graph$(trajectory) # Build a local connectedness graph from the observed trajectory
16: **for all** connections $c \in \mathcal{G}$ **do**
17: get $s_{start}, s_{end}, j', r'$ from c
18: $\mathcal{Q}(s_{start}, j'|a) \leftarrow td\_update\_skip(\mathcal{Q}(s_{start}, j'|a), r', s_{end})$ # See Equation 6
19: **end for**
20: **until** episode finished
21: **end for**

B. Implementation Details

Algorithm 1 details how to train a TEMPORL $\mathcal{Q}$-learning agent. All elements that are new to TEMPORL are shown in black whereas vanilla $\mathcal{Q}$-learning code is greyed out. The functions td_update (Line 12) and td_update_skip (Line 18) are formally stated in Equations 5 and 6 respectively and give the temporal difference updates required during learning.

$$\mathcal{Q}(s_t,a_t) = \mathcal{Q}(s_t,a_t) + \alpha \underbrace{\left(\underbrace{\left(r_t + \gamma \max \mathcal{Q}(s_{t+1},\cdot) \right)}_{\text{TD-Target}} - \mathcal{Q}(s_t,a_t) \right)}_{\text{TD-Delta}} \tag{5}$$

$$\mathcal{Q}(s_t,j_t|a_t) = \mathcal{Q}(s_t,j_t|a_t) + \alpha \underbrace{\left(\underbrace{\left(\sum_{k=0}^{j-1} \gamma^k r_{t+k} + \gamma^j \max \mathcal{Q}(s_{t+j},\cdot) \right)}_{\text{TD-Target}} - \mathcal{Q}(s_t,j_t|a_t) \right)}_{\text{TD-Delta}} \tag{6}$$

Where α is the learning rate and γ the discount factor. Note that the TD-Target in Equation 6 (as well as the skip $\mathcal{Q}$-function in Equation 4) is using the behaviour $\mathcal{Q}$-function and not the skip $\mathcal{Q}$-function. Thus, the skip $\mathcal{Q}$-function estimates the expected future rewards, assuming that the current skip will be the only skip in the MDP. This allows us to avoid overestimating $\mathcal{Q}$-values through multiple skips and focuses on learning of the value of the executed skip similar to double $\mathcal{Q}$-learning (van Hasselt, 2010). Further, learning of the skip-values does not interfere with learning of the behaviour $\mathcal{Q}$-function.

The function $build\_connectedness\_graph$ (Line 15) builds takes an observed trajectory and builds connectedness graph of states that are reachable by repeatedly playing the same action (see Figure 1 in the main paper). Each connection contains information about start and end states, the length of the skip and the discounted reward for that skip.

C. Used Compute Resources

Tabular & Deep RL Experiments on Featurized Environments For the tabular as well as the deep experiments on featurized environments, we evaluated all agents on a compute cluster with nodes equipped with two Intel Xeon Gold 6242 32-core CPUs, 20 MB cache and and 188GB (shared) RAM running Ubuntu 18.04 LTS 64 bit. In all cases, the agents were allocated one CPU. The tabular agents required at most 20 minutes to complete training, whereas the deep agents required at most 15 hours.

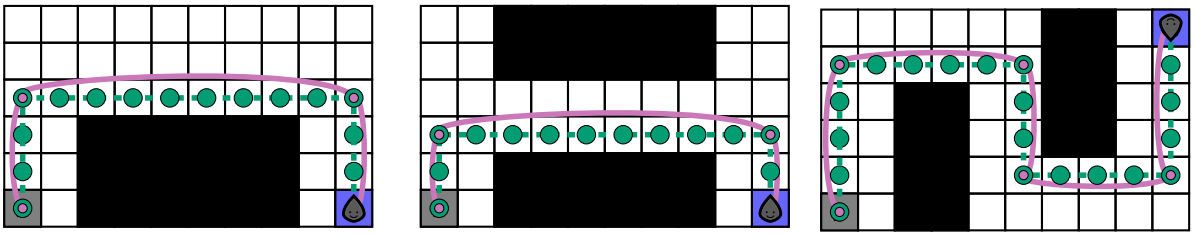
Deep RL Experiments on Atari Environments These experiments were run on a compute cluster with nodes equipped with two Intel Xeon E5-2630v4 and 128GB memory running CentOS 7. For training, the agents were allocated 10 CPUs and required at most 48 hours to complete training.

D. Gridworld Details

All considered environments (see Figure D1) are discrete, deterministic, have sparse rewards and have size 6×10. Falling off a cliff results in a negative reward (-1) and reaching a goal state results in a positive reward ($+1$). Both cliff and goal states terminate an episode. All other states result in no reward. An agent can only execute the actions up, $down$, $left$, $right$ with diagonal moves not possible. If the agent does not reach a goal/cliff in 100 steps, an episode terminates without a reward.

For the Cliff environment, a shortest path through the environment requires 16 steps. However, to reach the goal, decisions about unique actions are only required at 3 time points. The first is in the starting state and determines that action up should be repeated 3-times, the next is repeating action $right$ 10-times and the final one is repeating action $down$ 3-times. Thereby, an optimal proactive policy that is capable of joint decision of action and skip length requires $\sim 80\%$ fewer decisions than an optimal reactive policy that has to make decisions in each state. As the Bridge environment is very similar, but has a smaller cliff area below, an optimal proactive policy also requires roughly $\sim 80\%$ fewer decisions.

On the more complex ZigZag environment, an optimal policy requires 20 steps in total to reach the goal. In this environment however, an agent has to switch direction more often. Leading to a total of 5 required decisions. Thus in this environment an optimal proactive policy requires roughly 75% fewer decisions.



(a) Cliff (b) Bridge (c) ZigZag

Figure D1. Copy of Figure 4 form the main paper. 6×10 Grid Worlds. Agents have to reach a fixed goal state from a fixed start state. Large/small dots represent decision steps of vanilla and TEMPORL $\mathcal{Q}$-learning policies.

Table E1. Normalized AUC for reward and average number of decision steps for varying maximal skip-lengths J. All agents are trained with the same ϵ schedule. $\mathcal{R}$ denotes normalized area under the reward curve and D the average number of decision steps. Values are results of running 10 random seeds. Columns 1 and 7 are equivalent to columns 5 & 6 in Table 1.

(a) linear decaying ϵ-schedule

	$\mathcal{Q}$	t-$\mathcal{Q}$														
J	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\mathcal{R}$	0.57	0.63	0.76	0.87	0.93	0.93	0.92	0.91	0.90	0.91	0.88	0.87	0.86	0.87	0.85	0.84
D	83.6	36.5	20.6	13.2	10.1	8.3	7.7	7.8	7.5	7.4	7.6	7.4	7.6	7.6	7.8	7.4

(b) logarithmic decaying ϵ-schedule

$\mathcal{R}$	0.90	0.91	0.93	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.95	0.96	0.95	0.95
D	35.6	21.7	14.9	11.6	9.5	8.6	6.4	6.3	6.5	5.9	6.1	6.2	7.0	6.8	7.0	6.0

(c) constant $\epsilon = 0.1$

$\mathcal{R}$	0.95	0.98	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98
D	27.6	15.8	12.0	9.1	8.2	7.8	6.8	6.9	6.7	7.1	6.6	7.2	6.2	6.5	7.0	6.9

E. Influence of the Maximum Skip-Length

The maximum skip length J is a crucial hyperparameter of TEMPORL. A too large value might lead to many irrelevant choices which the agent has to learn to ignore; whereas a too small value might not reduce the complexity of the environment sufficiently enough, leading to barely an improvement over the vanilla counterpart. To evaluate the influence of the hyperparameter on our method we trained various TEMPORL agents with varying maximal skip-lengths, starting from 2 up to 16. Larger skips than 10 will never be beneficial for the agent as the agent is guaranteed to run into a wall for some steps. Depending on where in the environment the agent is located, smaller skip-values might allow it to quickly traverse through the environment.

Table E1 shows the influence of J on the ZigZag environment (see Figure 4c). In this environment, the largest skip value that is possible without running into a wall is 6. Thus, small skip values up to 5 quickly improve the performance over the vanilla counterpart, not only in terms of anytime performance but also in terms of required decisions. In the case of a suboptimal exploration policy, in the form of linearly decaying ϵ-greedy schedule (see Table E1a), larger skip-values quickly lead to a decrease in anytime performance, as the agent has to learn to never choose many non-improving skip actions.

For a more suiting exploration policy, too large skip-actions do not as quickly degrade the anytime performance of our TEMPORL agents. In the case of a logarithmically decaying ϵ schedule (Table E1b), we can see that skip sizes larger or equal than 12 start to negatively influence the anytime performance, whereas with a constant ϵ schedule only a skip-size of 16, nearly 3 times as large as the largest sensible choice, has a negative effect.

Similar observations can be made for deep TEMPORL on both Pedulum, MountainCar and LunarLander, see Tables 2 - 4 in the main paper. We can see that choosing larger maximal skip-values is beneficial, up to a point, at which many irrelevant, and potentially useless choices are in the action space. For these, TEMPORL first has to learn on which part of the skip-action-space to focus before really learning when new decisions need to be taken.

It is worth noting that, in the tabular case, all evaluated skip-sizes J result in better anytime-performance and a lower number of required decision points compared to vanilla $\mathcal{Q}$-learning, for all considered exploration strategies. In future work, we will study how to allow TEMPORL to select large skip-actions without needing to learn to distinguish between many irrelevant choices. One possible way of doing this could be by putting the skip-size on a log scale. For example using $\log_2$ could result in only 10 actions where a TEMPORL agent could skip up to 1024 steps ahead but would still be able to exert fine control with the smaller actions.

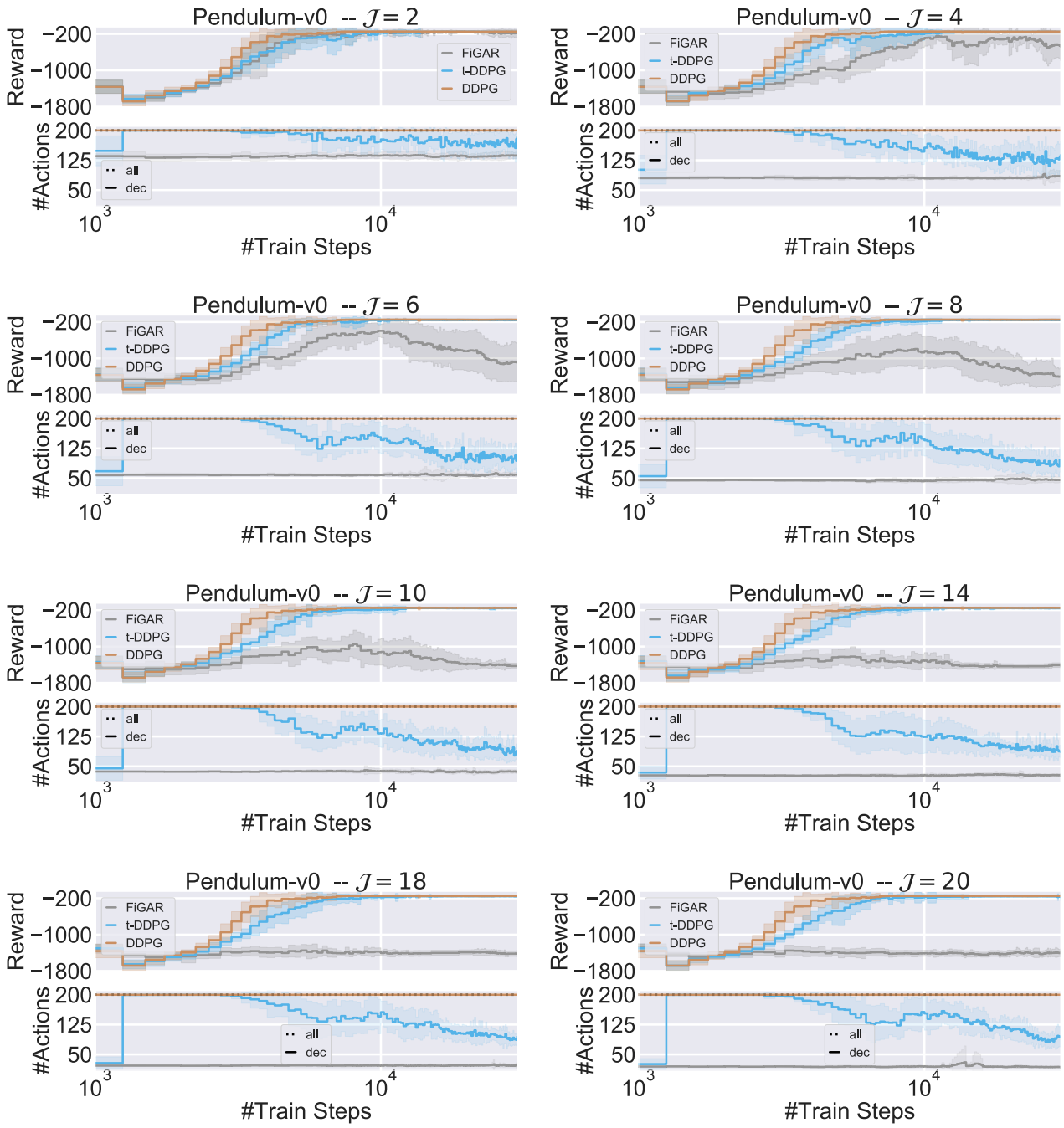


Figure F1. Learning curves of different DDPG agents on Pendulum-v0. J indicates the maximal skip-length used when training t-DDPG and FiGAR. Solid lines give the mean and the shaded area the standard deviation over 15 seeds. Top-row images show the reward achieved and bottom-row images the required steps and decisions per evaluation rollout.

F. DDPG Implementation Details and Additional Results

As base implementation for DDPG, we used publicly available code[5] and used the default hyperparameters, except we replaced the number of maximal training steps and initial random steps as described in the main paper. When implementing FiGAR, we followed the description by Sharma et al. (2017). Thus, the repetition policy uses a constant epsilon-greedy exploration. Likewise, we use a constant epsilon-greedy exploration to learn our t-DDPG.

For our t-DDPG implementation we could use the same algorithm as described in Algorithm 1. Only the greyed out parts of normal $\mathcal{Q}$-learning have to replaced by DDPG training specific elements. For example, for DDPG, the exploration policy for the actor is given by adding exploration noise rather than following an epsilon-greedy policy. Further, we again can make use of the base agents $\mathcal{Q}$-function as shown in Equation 6.

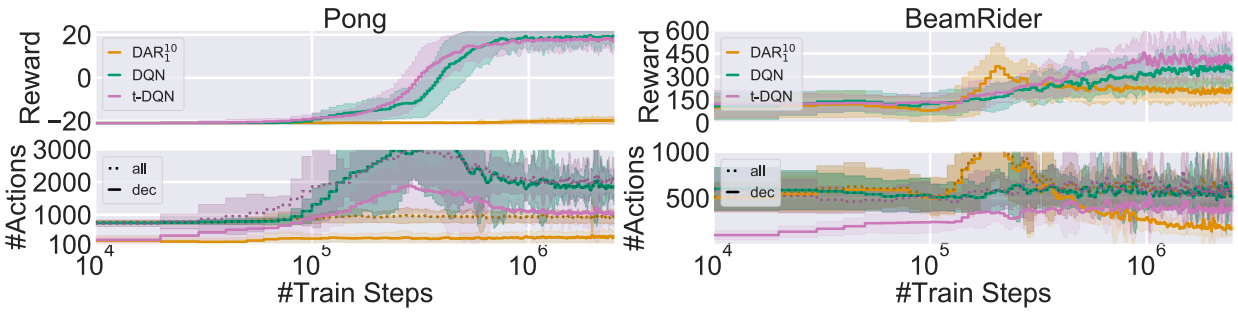
Figure F1 depicts the learning curve for all DDPG agents with increasing maximal skip-value. As described in the main paper, both FiGAR and t-DDPG slightly lag behind vanilla DDPG when only allowing for skips of length 2. However, with increasing max-skip value FiGAR quickly begins to struggle and in the end even converges to worse policies, always preferring large skip-values. Our t-DDPG using TEMPORL performs much more stable and is hardly affected by increasing the maximal skip length. Further, t-DDPG over time learns *when* it is necessary to switch to new actions, roughly halving the required decisions.

G. Featurized Environments Description

MountainCar is a challenging exploration task and requires an agent to control an under powered car to drive up a steep hill on one side (Moore, 1990). To reach the goal, an agent has to build up momentum. The agent always receives a reward of -1 until it has crossed the goal position and a reward of 0 afterwards. The observation consists of the car position and velocity and the agent can either accelerate to the left or right or do nothing. To build up momentum an agent potentially has to repeat the same action multiple times. Thus, we evaluate both t-DQN and DAR on the grid $\{2, 4, 6, 8, 10\}$ for the maximal (while keeping the minimal skip value fixed to 1) skip-value over 50 random seeds (see Tables 3a & 4a).

LunarLander The task for an agent is to land a space-ship on a lunar surface. To this end, the agent can choose to fire the main engine, steer left or right or do nothing. Firing of the engines incurs a small cost of -0.3, whereas crashing or successfully landing results in a large cost or reward of -100 and 100 respectively. We expect that an environment with such a dense reward, where actions directly influence the achieved reward does not benefit from leveraging skips.

[5] https://github.com/sfujim/TD3



(a) Pong (b) BeamRider

Figure H1. Evaluation performance on Atari environments. Solid lines give the mean and the shaded area the standard deviation over 15 random seeds. (top) Achieved rewards. (bottom) Length of executed policy ($\cdots$) and number of decisions (—) made by the policies.

H. Atari

Architectures *DQN:* As architecture for DQN we used that of Mnih et al. (2015) and used this as basis for our shared architecture. This architecture has three layers of convolutions to handle the 84×84 input images. The first convolution layer has 84 input channels, 32 output channels, a kernel size of 8 and a stride of 4. The second has 32 input channels, 64 output channels, a kernel size of 4 and a stride of 2. The second has 64 input channels, 64 output channels, a kernel size of 3 and a stride of 1. This is followed by two hidden layers with 512 units each.

TEMPORL: The shared architecture used by our TEMPORL agent uses the same architecture as just described but has an additional output stream for the skip-outputs. The skip output stream combines a hidden layer with 10 units together with the output of the last convolutional layer. It then processes these features again in two fully connected hidden layers with 512 units each.

DAR: Similarly, the DAR agent builds on the DQN architecture of Mnih et al. (2015). However, the final output layer is duplicated and the duplicate outputs act at a different time-resolution. To give DAR the same coarse control as would be possible with our TEMPORL agent we fix the fine and coarse control levels to 1 and 10 respectively.

Table H1. Hyperparameters used for the Atari Experiments

Hyperparameter	Value
Batch Size	32
γ	0.99
Gradient Clip	40.0
Target update frequency	500
Learning starts	10 000
Initial ϵ	1.0
Final ϵ	0.01
ϵ time-steps	200 000
Train frequency	4
Loss Function	Huber Loss
Optimizer	Adam
Learning rate	10^-4
β_1	0.9
β_2	0.999
Replay-Buffer Size	5×10^4
Skip Replay-Buffer Size	5×10^4
J	10

Additional Results on PONG: Our learned t-DQN exhibits a slight improvement in learning speed, PONG before being caught up by DQN (similar to the results on MsPacman in the main paper, see Figure 7a), with both methods converging to the same final reward. Nevertheless, TEMPORL learns to make use of different degrees of fine and coarse control to achieve the same performance, requiring roughly 1 000 fewer decisions.

The DAR agent really struggles to learn a meaningful policy on this game, never learning to properly avoid getting scored on or scoring itself. A likely reason for the poor performance is the choice of hyperparameters. Potentially choosing smaller skip-value for the coarse control could allow to learn better behaviour with DAR.

Additional Results on BEAMRIDER: Figure H1b shows an immediate benefit to jointly learning *when* and *how* to act through TEMPORL. Our t-DQN begins to learn faster and achieve a better final reward than vanilla DQN.

Interestingly, the DAR agent, starting out with choosing to mostly apply fine control starts to learn much faster than vanilla DQN and our TEMPORL agent, nearly reaching the final performance of vanilla DQN already $\approx$ 900 000 time-steps earlier. However, the performance starts to drop when DAR starts to increase usage of the coarse control. Once the DAR agents have learned this over-reliance on the coarse control, they do not recover, resulting in the worst final performance.

Appendix G

Appendix for Self-Paced Context Evaluation for Contextual Reinforcement Learning

Self-Paced Context Evaluation for Contextual Reinforcement Learning

Theresa Eimer [1] **André Biedenkapp** [2] **Frank Hutter** [2 3] **Marius Lindauer** [1]

A. Instance Sampling

AntGoal We uniformly sampled 100 different goals at a distance of at most 750 in both x- and y-direction for both training and test set respectively.

BallCatching The distance and goal coordinates were sampled uniformly for both training and test set. The distance ranged between $0.125 \cdot \pi$ and $0.5 \cdot \pi$, the x-coordinate between 0.6 and 1.1 and the y-coordinate between 0.75 and 4.0. Each instance set contains 100 instances.

PointMass For PointMass, we sampled two different instance sets. First, we used the context bounds of [-4, 4] for the goal position, [0.5, 8] for the goal width and [0, 4] for friction to uniformly sample instances. The goal was to cover the instance space as well as possible. Our second instance set was sampled using the target distribution of SPDRL, which are normal distributions for each context component with means 2.5, 0.5 and 0 respectively as well as standard deviations of 0.004, 0.00375, and 0.002.

B. Experiment Hardware & Hyperparameters

Hardware All experiments with SPACE and the baseline round robin agent were conducted on a slurm CPU cluster (see Table 1). The upper memory limit for these experiments was 1GB per run. The SPDRL experiments were replicated on a slurm GPU cluster consisting of 6 nodes with eight RTX 2080 Ti each. Here maximum memory was 10GB. Slurm scripts for the experiments on PointMass and Ant are provided in the supplementary material. Gridworld experiments are every small and can therefore be found in a jupyter notebook.

Machine no.	CPU model	cores	RAM
1	Xeon E5-2670	16	188 GB
2	Xeon E5-2680 v3	24	251
3-6	Xeon E5-2690 v2	20	125 GB
7-10	Xeon Gold 5120	28	187

Table 1: CPU cluster used for training

CartPole We used a DQN implementation in the top-10 on the environment leaderboard to ensure fair performance for round robin and SPACE agents (Chauhan, 2019). We did not change any hyperparameters from that implementation and used $\kappa = 1$ and $\eta = 2.5\%$ for all experiments.

Other benchmarks For both experiments we used stable baselines version 2.9.0 (Hill et al., 2018) with TRPO for PointMass and PPO2 for all other benchmarks. The policies are encoded by an MLP in both cases, with two layers of 64 units for PPO. For PointMass, we used the default from the SDPRL paper with 21 layers of 64 units each. The discount factor was 0.95. The PPO2 specfic hyperparameters included no gradient clipping, a GAE hyperparameter λ value of 0.99 and an entropy coefficient of 0. For TRPO we used again used the same hyperparameters as SPDRL with a GAE hyperparameter λ of 0.99, a maximum KL-Divergence of 0.004 and value function step size of around 0.24. Any hyperparameters not mentioned were left at the stable baselines' default values. The random seeds were used to seed the environments with the corresponding seeding method.

C. Additional Comparison to SPDRL

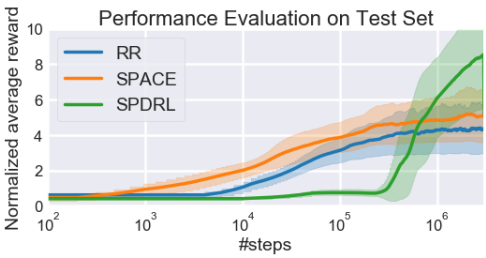


Figure 1: Mean reward per episode on a test set of hard instances with small goals and low friction.

In contrast to SPACE, SPDRL is designed to solve hard instances. To this end, it samples harder and harder instances over time. Therefore, we additionally study how SPACE, round robin (RR) and SPDRL compare on hard instances sampled from the SPDRL target distribution, see Figure 1. Instances in this distribution typically have small goal sizes and low friction, both of which contribute significantly to an increased difficulty.

As in the original paper, SPDRL was allowed to sample as many instances as needed from the distribution, whereas SPACE and RR still only got access to a finite set of 100 instances. In this setting, agents trained either via SPACE

or RR exhibit a similar learning behaviour as on the space covering instance set. For the first $\sim$ 200 000 steps both agents outperform the agent trained via SPDRL; RR anyway focuses on the whole target distribution from the beginnig and SPACE is more free in the way it can select instances with fast training progress. During this time, SPDRL trains the agents on some easy instances, while gradually adapting the instance distribution to focus on ever more difficult tasks. Note that the level of difficulty is not determined solely by the agent being trained via SPDRL, as done in SPACE, but is determined by an expert beforehand.

Once the agent trained via SPDRL is capable of homing in on the difficult instances it outperforms the other agents, as it can exploit its domain knowledge to sample ever more similarly difficult instances, while SPACE and RR are stuck with the limited number of example instances and still try to cover the entire instance space. To achieve this feat, SPDRL requires substantial expert knowledge about which instances to focus on. In essence, the agent trained via SPDRL in the end is only capable of solving a few hard instances with very little variation and will fail to perform well on instances that are not narrowly aligned with the assumed instance distribution.

To be able to know which instances SPDRL should focus on, additional time and effort have to be spent to identify how to quantify *difficulty* for SPDRL. This effort is not reflected in Figure 1 and would move the curve of SPDRL even further to the right.

D. Does the Training Set Size Matter?

To answer this question, we used SPACE to train agents with varying instance set sizes. Figure 2 shows the test performance for differently sized instance sets. Intuitively, one might think that performance should improve with more instances as they cover the instance space better. Indeed, the results for training sets with only 25 and 50 instances are visibly worse than for larger sets. On the remaining instance sets, the agent show very similar performance, however. Note that the performance seems to increase from an instance set size of 100 to 200, but slightly drops again afterwards. There are multiple factors potentially contributing to this effect.

The first is that the agent cannot incorporate any more information from the additional instances, maybe due to limited network capacity or due to the fact that smaller instance sets already cover the space adequately. Furthermore, as we only extend the instance set by one instance at a time, there are more learning steps between curriculum iterations the larger the instance set is, thereby slowing the process down. Especially an agent trained on 1 600 instances will suffer from this.

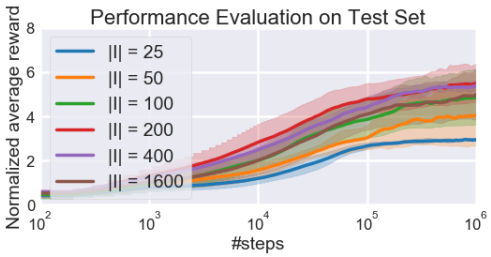


Figure 2: Mean reward per episode on test set for different sized instance sets.

Lastly, SPACE improves upon the RR baseline by ordering training instances and thus smoothing the progression through the instance space. Larger instance sets offer an inherently smoother representation of the instance distribution, therefore diminishing the effect of SPACE. In real-world application settings, we will rarely have access to such large numbers of instances and therefore, it is unlikely that such diminishing performance effects can be observed. This shows that the strength of our method comes to full effect when learning on a sparse representation of our instance space.

E. Comparison of SPACE Curricula

To give some insight into which curricula SPACE found on our benchmark environments, we compare how they behave across random seeds and how they compare to cSPACE curricula. We use Kendall's tau to determine how similar the order in which the instances are added to the training set is.

On PointMass, SPACE finds a curriculum that stay very consistent across all random seeds, showing a correlation of at least 98.9% each to the mean curriculum. The same is true for the cSPACE variation, where the correlation is above 93.8% per seed. Interestingly, these curricula are uncorrelated with a correlation of -0.04. In both we cannot make out a human readable progression in a single context feature (see Figure 3), their curricula do not correspond to any manual instance ordering. As both perform well nonetheless, we can see that learning can be improved by multiple different curricula on this environment.

SPACE and cSPACE produce almost equally unrelated curricula on AntGoal (correlation of 0.07), but while the curriculum stays as consistent across seeds for cSPACE, the same cannot be said for SPACE. Here the correlation to the average curriculum ranges from 14.1% to 52.4%. The correlations between the seed curricula fall into the same range, confirming that the SPACE agent trains on a very different curriculum for each seed. CartPole shows a similar behaviour, the curriculum varying quite a bit between

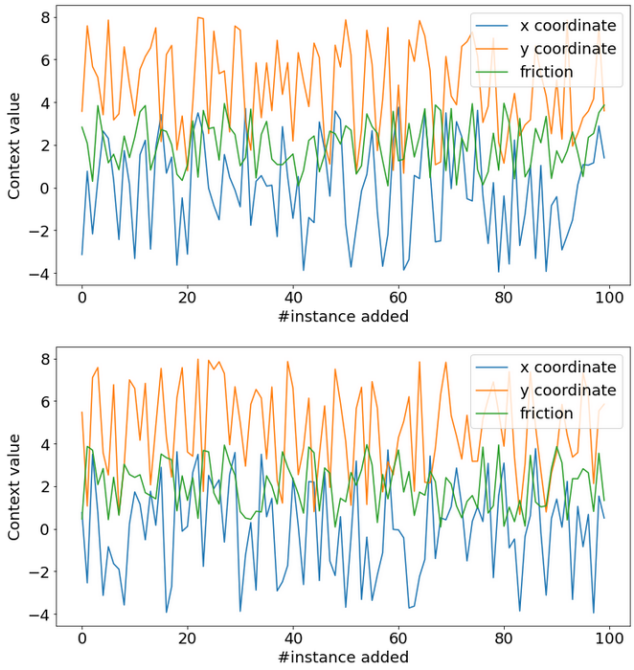


Figure 3: Context feature progression during training for SPACE curriculum (top) and cSPACE curriculum (bottom).

seeds. Therefore we can conclude that SPACE does not find a singular curriculum, but depends on the initialization of environment and model. This is in contrast to cSPACE which stays relatively static due to the context features being constant.

These comparisons suggest that we neither SPACE nor cSPACE finds an optimal curriculum for PointMass, AntGoal or CartPole. It seems, however, that we do not need an optimal curriculum for training at all, as even the 10 very different curricula SPACE finds on AntGoal perform vastly superior to the round robin default. Curriculum Learning should thus focus on reliably and quickly finding good curricula in addition to finding qualitatively better ones.

F. The Influence of Catastrophic Forgetting

When training across multiple instances, forgetting already learnt policies on a subset of instances is a concern (Beaulieu et al., 2020). We analyze how often SPACE and RR agents forget policy components in our PointMass experiments by observing performance development during training. We selected PointMass for this analysis as here policies that are diverse both in how they react to different goal settings and different friction levels are required. That means the policy has to completely change between the extremes of the context which is not required of our other benchmarks where underlying mechanics, e.g. walking for the Ant, stay very similar.

During the training on PointMass, we observed 8 out of 100 instances for which the performance decays after an initial improvement. We would expect the performance to stay at least constant if no forgetting takes place, so the agent likely forgets parts of the policy for these instances in favor of improving on others. The effect is about the same size for round robin agents where we can observe the same for 6 out of 100 instances.

Another reason for attributing this performance decay to forgetting is that on a purely goal-based PointMass variation, the number of instances on which we can observe this effect is slightly smaller (only 4 instances), though not significantly so. All performance decay happens after learning has stagnated on all instances, however. In this easier, purely goal-based setting we could therefore stop training early and would avoid performance decay entirely. This points towards the added complexity of the setting being harder to capture for our agents.

While the effects on both SPACE and RR agents are not very large in our experiments, catastrophic forgetting is therefore certainly important in the field of contextual RL. Future work could on integrate SPACE with existing efforts to reduce this effect like ANML (Beaulieu et al., 2020). A specific aspect of this research that would need to be extended is preventing forgetting in continuous context spaces in addition to the existing successes in discrete ones.

References

Beaulieu, S., Frati, L., Miconi, T., Lehman, J., Stanley, K. O., Clune, J., and Cheney, N. Learning to continually learn. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, 2020.

Chauhan, K. Cartpole_DQN. `https://github.com/kapilnchauhan77/CartPole_DQN`, 2019.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

Appendix H

Appendix for CARL: A Benchmark for Contextual and Adaptive Reinforcement Learning

A Benchmark Categories

To encourage generalization in RL, we chose a wide variety of common task characteristics as well as well-known environments as the basis of CARL.

The physical simulation environments (Brax, box2d and classic control) defining a dynamic body in a static world have similar context features like gravity, geometry of the moving body, position and velocity, mass, friction and joint stiffness. For brevity, we only detail the context features of CARLFetch and list all other environments' context features in Section G of the appendix.

CARLFetch embeds Brax' Fetch [17] as a cMDP, see Figure 1a. The goal of Fetch is to move the agent to the target area. The context features joint stiffness, gravity, friction, (joint) angular damping, actuator strength, torso mass as well as target radius and distance define the context. The defaults of the context features are copied from the original environment. Furthermore, appropriate bounds must be set for the specific application. We set the bounds such that the environment's purpose is not violated, e.g., restricting the gravity towards the ground greater than 0 (otherwise the agent would fly up and it would be impossible to act).

Besides physical simulation environments, CARL provides two more specific, challenging environments. The first is the CARLMarioEnv environment built on top of the TOAD-GAN level generator [3, 50]. It provides a procedurally generated game playing environment (similarly to the ones discussed in Section 4) that allows customization of the generation process. This environment is therefore especially interesting for exploring representation learning for the purpose of learning to better generalize. Secondly, we move closer to real-world application by including the CARLRNADesignEnvironment [48]. The task here is to design RNA sequences given structural constraints. As two different datasets of structures and their instances are used in this benchmark, it is ideally suited for testing policy transfer between RNA structures.

A.1 Pendulum's Dynamic Equations

Because we use gym's Pendulum [7] for our experiments Q1 and Q2 (see section 6), we provide the dynamic equations to show the simplicity of the system. The state consists of the angular position θ and velocity of the pendulum $\dot{\theta}$. The discrete equation defining the behavior of the environment is defined as follows:

$$\dot{\theta}_{k+1} = \dot{\theta}_k + (-\frac{3g}{2l}\sin(\theta_k + \pi) + \frac{3}{m \cdot l^2}u_k) \cdot \Delta t$$
$$\theta_{k+1} = \theta_k + \dot{\theta}_{k+1} \cdot \Delta t\,.$$

Here, k is the index of the iteration/step, g the gravity, l the length of the pendulum, u the control input and Δt the timestep.

B Hardware and Hyperparameters

Hardware All experiments on all benchmarks were conducted on a slurm CPU cluster if not stated otherwise (see Table 2). The experiments for CARLMarioEnv were replicated on a slurm GPU cluster consisting of 6 nodes with eight RTX 2080 Ti each.

Table 2: CPU cluster used for training

Machine no.	CPU model	cores	RAM
1	Xeon E5-2670	16	188 GB
2	Xeon E5-2680 v3	24	251
3-6	Xeon E5-2690 v2	20	125 GB
7-10	Xeon Gold 5120	28	187

Hyperparameters and Training Details We used agents from stable baselines 3 [42] (version 1.1.0) for all of our experiments. For the DQN (used in CARLLunarLanderEnv, Section 6.3) and the PPO agent (used in CARLMarioEnv in Section C) we employ the hyperparameters from the stable baselines zoo [41], see Table 3. For the DDPG agent (used for CARLPendulumEnv in Section 6.1

and 6.2) we use the defaults with a MLP policy. We train each agent for 10^6 steps. Every 5000 steps we evaluate one episode on each train instance and report the mean reward across instances. All experiments can be reproduced using the scripts we provide with the benchmark library at `https://www.github.com/automl/CARL`.

Table 3: Hyperparameters from stable baselines zoo [41] for the agents used. Blank fields mean default values from stable baselines agent.

Hyperparameter	DQN	PPO
n_envs	1	8
policy	MlpPolicy	CnnPolicy
n_steps		128
n_epochs		4
learning_rate	6.3e-4	lin_2.5e-4
batch_size	128	256
n_epochs	4	
buffer_size	50000	
learning_starts	0	
gamma	0.99	
target_update_interval	250	
train_freq	4	
gradient_steps	-1	
exploration_fraction	0.12	
exploration_final_eps	0.1	
clip_range		lin_0.1
policy_kwargs	net_arch=[256, 256]	
vf_coef		0.5
ent_coef		0.01

C Additional Experimental Results

To further illustrate the influence of varying context, we show experimental results for a PPO agent trained on the CARLMarioEnv. Again, the agent is trained for 5 different random seeds and with 100 training instances. In CARLMarioEnv different instances (Mario levels) are created by using TOAD-GAN [50]. By varying the noise input vector for TOAD-GAN we can generate different levels and the greater the noise, the greater the differences to the original level. Because CARLMarioEnv is pixel-based the context is implicitly coded in the state and we hide the context. As we can see in Figure 7 a diverse training distribution ($\sigma_{\text{rel}} = 0.2$) increases the performance and facilitates generalization. On the other hand if the noise becomes too large ($\sigma_{\text{rel}} = 0.5$) the performance decreases again. A reason might be that the levels for this noise level are very diverse and thus the current setup, with only implicit context, might not be suitable.

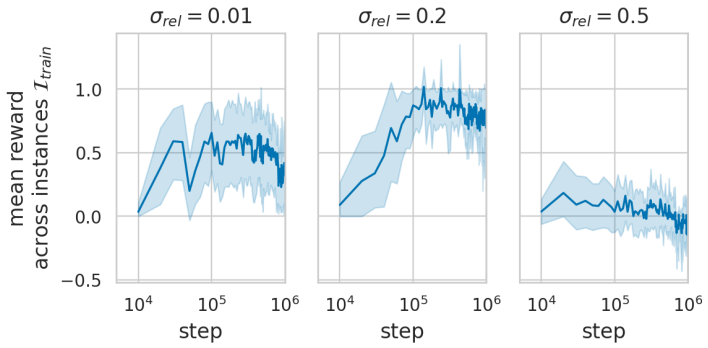


Figure 7: Training performance on CARLMarioEnv where only noise on the input generative vector is changed.

D Open Challenges Enabled by CARL

We used CARL to demonstrate the usefulness of a benchmark that can provide the ground truth of available context information. Based on that, we showed that making such information about the environment explicitly available to the agent enables faster training and transfer of agents (see Section 6). While this already provides valuable insights to the community that increasingly cares about learning agents capable of generalization (see Sections 1 & 3) CARL enables to study further open challenges for general RL.

D.1 Challenge I: Representation Learning

Our experiments using CARL demonstrated that an agent that is given access to context information is capable of learning better than an agent that has to learn behaviours given an implicit context via state observations. This provides evidence that disentangling the representation learning aspect from the policy learning task reduces complexity. As CARL provides a ground truth for representations of environment properties we envision future work on principled studies of novel RL algorithms that, by design, disentangle representation learning and policy learning (see, e.g., [44, 18, 62] as first works along this line of research). The ground truth given by the context would allow to measure the quality of learned representations and allows us to relate this to true physical properties of an environment.

Another use-case of CARL we envision under the umbrella of representation learning follows the work of environment probing policies [64]. There, exploratory policies are learned that allow to identify which environment type an agent encounters. This is complementary to the prior approaches as representations are not jointly learned with the behaviour policies as in the previously discussed approaches but rather in a separate offline phase. Based on CARL, huge amounts of meta-data could be collected that will enable the community to make use of classical meta-algorithmic approaches such as algorithm selection [46] for selecting previously learned policies or learning approaches.

D.2 Challenge II: Uncertainty of RL Agents

With the access to context information CARL enables to study the influence of noise on RL agents in a novel way. While prior environments enabled studies of the behaviour of agents when they could not be certain about their true state in a particular environment, CARL further allows to study agents behaviours in scenarios with uncertainty on their current contextual environment, e.g., because of noise on the context features. In practical deployment of RL, this is reasonable concern since context feature have to be measured somehow by potentially noisy censors. As this setting affects the overall transition dynamics, CARL provides a unique test-bed in which the influence of uncertainty can be studied and how RL agents can deal with such.

D.3 Challenge III: Continual Learning

With the flexibility and easy modifiability of CARLs provided contexts, CARL is suitable for studying continual reinforcement learning agents. In this setting, the distributions provided by CARL could be modified, e.g., gradually shifted, during the training procedure. For example, CARL could be used to evaluate the behaviour of an agent in the Brax environments where one or more joints become stiffer over time. A learning agent would need to be able to handle this and adapt its gait accordingly. In particular, one could at some point "repair" the agent and reset the joints to their original stiffness. This would then allow to evaluate whether the agent has "unlearned" the original gait. In the same way, CARL allows also to study how agents would react to spontaneous, drastic changes, e.g., broken legs or changes of the environment such as changes of weather conditions.

D.4 Challenge IV: Interpretable and Explainable Deep RL

Trust is a crucial factor, for which interpretability or explainability often is mandatory. With the provided ground truth through the explicit use of context features, CARL could be the base for studying interpretability and explainability of (deep) RL. By enabling AutoRL studies and different representation learning approaches, CARL will contribute to better interpret the training procedures.

CARL further allows to study explainability on the level of learned policies. We propose to study the sensitivity of particular policies to different types of context. Thus, the value and variability of a

context might serve as a proxy to explain the resulting learned behavior. Such insights might then be used to predict how policies might look like or act (e.g., in terms of frequency of action usage) in novel environments, solely based on the provided context features.

D.5 Challenge V: AutoRL

AutoRL (e.g., [22, 48, 21, 9]) addresses the optimization of the RL learning process. To this end, hyperparameters, architectures or both of agents are adapted either on the fly [22] or once at the beginning of a run [48]. However, as AutoRL typically requires large compute resources for this procedure, optimization is most often done only on a per-environment basis. It is reasonable to assume that such hyperparameters might not transfer well to unseen environments, as the learning procedures were not optimized to be robust or to facilitate generalization, but only to improve the reward on a particular instance.

As CARL provides easy-to-use contextual extensions of a diverse set of RL problems, it could be used to drive research in this open challenge of AutoRL. First of all, it enables a large scale-study to understand how static and dynamic configuration approaches complement each other and when one approach is to be preferred over another. Such a study will most likely also lead to novel default hyperparameter configurations that are more robust and tailored to fast learning and good generalization. In addition, it will open up the possibility to study whether it is reasonable to use a single hyperparameter configurations or whether a mix of configurations for different instances is required [59]. Furthermore, with the flexibility of defining a broad variety of instance distributions for a large set of provided context features, experiments with CARL would allow researchers to study which hyperparameters play a crucial role in learning general agents similar to studies done for supervised machine learning [56] or AI algorithms [6].

D.6 Challenge VI: High Confidence Generalization

The explicit context of the CARL benchmark enables tackling another challenge in the field of safe RL. High Confidence Generalization algorithms (HCGAs) [25] provide safety guarantees for the generalization of agents in testing environments. Given a worst-case performance bound, the agent can be tasked to decide whether a policy is applicable in an out-of-distribution context or not. This setting is especially important for the deployment of RL algorithms in the real world where policy failures can be costly and the context of an environment is often prone to change. CARL has the potential to facilitate the development of HCGAs that base their confidence estimates on the context of an environment.

E Future Maintenance

As our benchmark draws from several different RL environments as dependencies, we realize that it will need regular maintenance and updating. Furthermore, we would like to include more benchmarks and options that are closer to real-world applications. In part, we of course hope that the community will embrace CARL and work with us to extend it in order to match the needs of researchers working in cRL. We acknowledge, however, that relying on community driven progress only is infeasible. Therefore we commit to updating the current benchmark version including its dependencies at least twice a year or whenever critical updates in dependencies are released. As we plan to continue using GitHub for hosting, versioning as well as providing continued access to previous versions is feasible. We also aim to fix any issues that are brought to our attention in a reasonable timeframe. In case community-driven benchmarks are added, we will ensure the continued functionality of the benchmark as a whole (as far as our resources will allow). As we are researching solution methods in the field of cRL ourselves, we expect to contribute further benchmarks of our own as well.

F Statement

The authors' acknowledge that they bear all responsibility in case of violation of rights, etc., and confirmation of the data license.

G Context Features for Each Environment

We list all registered context features with their defaults, bounds and types for each environment family in Table 4 (classic control), Table 5 (box2d), Table 6 (brax) and Table 7 (RNA and Mario).

Table 4: Context Features: Defaults, Bounds and Types for OpenAI gym's Classic Control environments [7]

(a) CARLCartPoleEnv

Context Feature	Default	Bounds	Type
force_magnifier	10.00	(1, 100)	int
gravity	9.80	(0.1, inf)	float
masscart	1.00	(0.1, 10)	float
masspole	0.10	(0.01, 1)	float
pole_length	0.50	(0.05, 5)	float
update_interval	0.02	(0.002, 0.2)	float

(b) CARLPendulumEnv

Context Feature	Default	Bounds	Type
dt	0.05	(0, inf)	float
g	10.00	(0, inf)	float
l	1.00	(1e-06, inf)	float
m	1.00	(1e-06, inf)	float
max_speed	8.00	(-inf, inf)	float

(c) CARLMountainCarEnv

Context Feature	Default	Bounds	Type
force	0.00	(-inf, inf)	float
goal_position	0.50	(-inf, inf)	float
goal_velocity	0.00	(-inf, inf)	float
gravity	0.00	(0, inf)	float
max_position	0.60	(-inf, inf)	float
max_speed	0.07	(0, inf)	float
min_position	-1.20	(-inf, inf)	float
start_position	-0.50	(-1.5, 0.5)	float
start_position_std	0.10	(0.0, inf)	float
start_velocity	0.00	(-inf, inf)	float
start_velocity_std	0.00	(0.0, inf)	float

(d) CARLAcrobotEnv

Context Feature	Default	Bounds	Type
link_com_1	0.50	(0, 1)	float
link_com_2	0.50	(0, 1)	float
link_length_1	1.00	(0.1, 10)	float
link_length_2	1.00	(0.1, 10)	float
link_mass_1	1.00	(0.1, 10)	float
link_mass_2	1.00	(0.1, 10)	float
link_moi	1.00	(0.1, 10)	float
max_velocity_1	12.57	(1.26, 125.66)	float
max_velocity_2	28.27	(2.83, 282.74)	float

Table 5: Context Features: Defaults, Bounds and Types for OpenAI gym's Box2d environments [7]

(a) CARLBipedalWalkerEnv

Context Feature	Default	Bounds	Type
FPS	50.00	(1, 500)	float
FRICTION	2.50	(0, 10)	float
GRAVITY_X	0.00	(-20, 20)	float
GRAVITY_Y	-10.00	(-20, -0.01)	float
INITIAL_RANDOM	5.00	(0, 50)	float
LEG_DOWN	-0.27	(-2, -0.25)	float
LEG_H	1.13	(0.25, 2)	float
LEG_W	0.27	(0.25, 0.5)	float
LIDAR_RANGE	5.33	(0.5, 20)	float
MOTORS_TORQUE	80.00	(0, 200)	float
SCALE	30.00	(1, 100)	float
SPEED_HIP	4.00	(1e-06, 15)	float
SPEED_KNEE	6.00	(1e-06, 15)	float
TERRAIN_GRASS	10.00	(5, 15)	int
TERRAIN_HEIGHT	5.00	(3, 10)	float
TERRAIN_LENGTH	200.00	(100, 500)	int
TERRAIN_STARTPAD	20.00	(10, 30)	int
TERRAIN_STEP	0.47	(0.25, 1)	float
VIEWPORT_H	400.00	(200, 800)	int
VIEWPORT_W	600.00	(400, 1000)	int

(b) CARLLunarLanderEnv

Context Feature	Default	Bounds	Type
FPS	50.00	(1, 500)	float
GRAVITY_X	0.00	(-20, 20)	float
GRAVITY_Y	-10.00	(-20, -0.01)	float
INITIAL_RANDOM	1000.00	(0, 2000)	float
LEG_AWAY	20.00	(0, 50)	float
LEG_DOWN	18.00	(0, 50)	float
LEG_H	8.00	(1, 20)	float
LEG_SPRING_TORQUE	40.00	(0, 100)	float
LEG_W	2.00	(1, 10)	float
MAIN_ENGINE_POWER	13.00	(0, 50)	float
SCALE	30.00	(1, 100)	float
SIDE_ENGINE_AWAY	12.00	(1, 20)	float
SIDE_ENGINE_HEIGHT	14.00	(1, 20)	float
SIDE_ENGINE_POWER	0.60	(0, 50)	float
VIEWPORT_H	400.00	(200, 800)	int
VIEWPORT_W	600.00	(400, 1000)	int

(c) CARLVehicleRacingEnv

Context Feature	Default	Bounds	Type
VEHICLE	0	-	categorical, $n = 29$

Table 6: Context Features: Defaults, Bounds and Types for Google Brax environments [17]

(a) CARLAnt

Context Feature	Default	Bounds	Type
actuator_strength	300.00	(1, inf)	float
angular_damping	-0.05	(-inf, inf)	float
friction	0.60	(-inf, inf)	float
gravity	-9.80	(-inf, -0.1)	float
joint_angular_damping	35.00	(0, inf)	float
joint_stiffness	5000.00	(1, inf)	float
torso_mass	10.00	(0.1, inf)	float

(b) CARLFetch

Context Feature	Default	Bounds	Type
actuator_strength	300.00	(1, inf)	float
angular_damping	-0.05	(-inf, inf)	float
friction	0.60	(-inf, inf)	float
gravity	-9.80	(-inf, -0.1)	float
joint_angular_damping	35.00	(0, inf)	float
joint_stiffness	5000.00	(1, inf)	float
target_distance	15.00	(0.1, inf)	float
target_radius	2.00	(0.1, inf)	float
torso_mass	1.00	(0.1, inf)	float

(c) CARLGrasp

Context Feature	Default	Bounds	Type
actuator_strength	300.00	(1, inf)	float
angular_damping	-0.05	(-inf, inf)	float
friction	0.60	(-inf, inf)	float
gravity	-9.80	(-inf, -0.1)	float
joint_angular_damping	50.00	(0, inf)	float
joint_stiffness	5000.00	(1, inf)	float
target_distance	10.00	(0.1, inf)	float
target_height	8.00	(0.1, inf)	float
target_radius	1.10	(0.1, inf)	float

(d) CARLHumanoid

Context Feature	Default	Bounds	Type
angular_damping	-0.05	(-inf, inf)	float
friction	0.60	(-inf, inf)	float
gravity	-9.80	(-inf, -0.1)	float
joint_angular_damping	20.00	(0, inf)	float
torso_mass	8.91	(0.1, inf)	float

Table 7: Context Features: Defaults, Bounds and Types for RNA Design [48] and Mario Environment [3, 50]

(a) CARLRnaDesignEnv

Context Feature	Default	Bounds	Type
mutation_threshold	5	(0.1, inf)	float
reward_exponent	1	(0.1, inf)	float
state_radius	5	(1, inf)	float
dataset	eterna	-	categorical, $n = 3$
target_structure_ids	f(dataset)	(0, inf)	list of int

(b) CARLMarioEnv

Context Feature	Default	Bounds	Type
level_index	0	-	categorical, $n = 15$
noise	f(level_index, width, height)	(-1, 1)	float
mario_state	0	-	categorical, $n = 3$
mario_inertia	0.89	(0.5, 1.5)	float

Bibliography

Adriaensen, S., A. Biedenkapp, G. Shala, N. Awad, T. Eimer, M. Lindauer, and F. Hutter (2022). “Automated Dynamic Algorithm Configuration”. In: *arXiv:2205.13881 [cs.AI]* (cit. on pp. 17, 39).

Almeida, D., C. Winter, J. Tang, and W. Zaremba (2021). “A Generalizable Approach to Learning Optimizers”. In: *arXiv:2106.00958 [cs.LG]* (cit. on p. 19).

Ansótegui, C., M. Sellmann, and K. Tierney (2009). “A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms”. In: *Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP’09)*. Ed. by I. Gent. Vol. 5732. Lecture Notes in Computer Science. Springer, pp. 142–157 (cit. on p. 189).

Awad, N., G. Shala, D. Deng, N. Mallik, M. Feurer, K. Eggensperger, A. Biedenkapp, D. Vermetten, H. Wang, C. Doerr, M. Lindauer, and F. Hutter (2020). “Squirrel: A Switching Hyperparameter Optimizer Description of the entry by AutoML.org & IOHprofiler to the NeurIPS 2020 BBO challenge”. In: *arXiv:2012.08180 [cs.LG]* (cit. on pp. 17, 18).

Bach, F. R. and E. Moulines (2011). “Non-Asymptotic Analysis of Stochastic Approximation Algorithms for Machine Learning”. In: *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS’11)*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger. Curran Associates, pp. 451–459 (cit. on pp. 9, 19).

Bellemare, M. G., Y. Naddaf, J. Veness, and M. Bowling (2013). “The Arcade Learning Environment: An Evaluation Platform for General Agents”. In: *jair* 47, pp. 253–279 (cit. on pp. 15, 24).

Bellman, R. (1957). “A Markovian decision process”. In: *Journal of Mathematics and Mechanics*, pp. 679–684 (cit. on p. 22).

Bellman, R. (1957/2003). *Dynamic Programming*. Dover Publications. ISBN: 0-486-42809-5 (cit. on pp. 23, 24). Original: *Dynamic Programming*. Princeton University Press, 1957.

Benjamins, C., T. Eimer, F. Schubert, A. Biedenkapp, B. Rosenhan, F. Hutter, and M. Lindauer (2021). “CARL: A Benchmark for Contextual and Adaptive Reinforcement Learning”. In: *Workshop on Ecological Theory of Reinforcement Learning (EcoRL@NeurIPS’21)* (cit. on pp. 17, 163).

Benjamins, C., T. Eimer, F. Schubert, A. Mohan, A: Biedenkapp, B. Rosenhan, F. Hutter, and M. Lindauer (2022). “Contextualize Me – The Case for Context in Reinforcement Learning”. In: *arXiv:2202.04500 [cs.LG]* (cit. on p. 188).

Bertsekas, D. P. and J. N. Tsitsiklis (1996). *Neuro-dynamic programming*. Vol. 3. Optimization and neural computation series. Athena Scientific. ISBN: 1886529108 (cit. on p. 22).

Biedenkapp, A., H. Bozkurt, T. Eimer, F. Hutter, and M. Lindauer (June 2020). “Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework”. In: *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI’20)*. Ed. by J. Lang, G. De Giacomo, B. Dilkina, and M. Milano, pp. 427–434 (cit. on pp. 16, 29).

Biedenkapp, A., N. Dang, M. S. Krejca, F. Hutter, and C. Doerr (2022a). “Theory-inspired Parameter Control Benchmarks for Dynamic Algorithm Configuration”. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO’22)*. Ed. by J. Fieldsend. ACM (cit. on pp. 17, 18, 125).

Biedenkapp, A., J. Marben, M. Lindauer, and F. Hutter (2018). "CAVE: Configuration Assessment, Visualization and Evaluation". In: *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*. Ed. by R. Battiti, M. Brunato, I. Kotsireas, and P. Pardalos. Lecture Notes in Computer Science. Springer (cit. on p. 18).

Biedenkapp, A., R. Rajan, F. Hutter, and M. Lindauer (2021). "TempoRL: Learning When to Act". In: *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 914–924 (cit. on pp. 16, 21, 139).

Biedenkapp, A., D. Speck, S. Sievers, F. Hutter, M. Lindauer, and J. Seipp (2022b). "Learning Domain-Independent Policies for Open List Selection". In: *Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL@ICAPS'22)* (cit. on p. 188).

Biere, A., M. Heule, H. van Maaren, and T. Walsh, eds. (2009). *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press. ISBN: 978-1-58603-929-5 (cit. on p. 3).

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba (2016). "OpenAI Gym". In: *arXiv:1606.01540 [cs.LG]* (cit. on p. 14).

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020). "Language Models are Few-Shot Learners". In: (cit. on p. 188).

Campbell, M., A. J. Hoane Jr., and F. Hsu (2002). "Deep Blue". In: *Artificial Intelligence* 134.1-2, pp. 57–83 (cit. on p. 3).

Chrabaszcz, P., I. Loshchilov, and F. Hutter (2018). "Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. Ed. by J. Lang. ijcai.org, pp. 1419–1426 (cit. on p. 23).

Daniel, C., J. Taylor, and S. Nowozin (2016). "Learning Step Size Controllers for Robust Neural Network Training". In: *Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16)*. Ed. by D. Schuurmans and M. Wellman. AAAI Press (cit. on p. 19).

Dimopoulos, Y., B. Nebel, and J. Koehler (1997). "Encoding Planning Problems in Nonmonotonic Logic Programs". In: *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97*. Ed. by S. Steel and R. Alami. Vol. 1348. Lecture Notes in Computer Science. Springer, pp. 169–181 (cit. on p. 3).

Doerr, B. (2019). "Analyzing randomized search heuristics via stochastic domination". In: *Theoretical Computer Science* 773, pp. 115–137. URL: https://doi.org/10.1016/j.tcs.2018.09.024 (cit. on p. 14).

Doerr, B. and C. Doerr (2018). "Theory of Parameter Control for Discrete Black-Box Optimization: Provable Performance Gains Through Dynamic Parameter Choices". In: *arXiv:1804.05650 [cs.NE]* (cit. on p. 14).

Eimer, T., A. Biedenkapp, F. Hutter, and M. Lindauer (2021a). "Self-Paced Context Evaluation for Contextual Reinforcement Learning". In: *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 2948–2958 (cit. on pp. 16, 151).

Eimer, T., A. Biedenkapp, M. Reimer, S. Adriaensen, F. Hutter, and M. Lindauer (2021b). "DACBench: A Benchmark Library for Dynamic Algorithm Configuration". In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI'21*. Ed. by Z. Zhou. ijcai.org, pp. 1668–1674 (cit. on pp. 16, 117).

Feurer, M., A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter (2019). "Auto-sklearn: Efficient and Robust Automated Machine Learning". In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 5. The Springer Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer. Chap. 6, pp. 113–134 (cit. on p. 189).

François-Lavet, V., R. Fonteneau, and D. Ernst (2015). "How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies". In: *arXiv:1512.02011 [cs.LG]* (cit. on pp. 22, 188).

Franke, J. KH, G. Köhler, A. Biedenkapp, and F. Hutter (2021). "Sample-Efficient Automated Deep Reinforcement Learning". In: *Proceedings of the International Conference on Learning Representations (ICLR'21)*. Published online: iclr.cc (cit. on p. 17).

Ghallab, M., D. S. Nau, and P. Traverso (2004). *Automated planning - Theory and Practice*. Elsevier. ISBN: 978-1-55860-856-6 (cit. on p. 3).

Goodfellow, I. J., Y. Bengio, and A. C. Courville (2016). *Deep Learning*. Adaptive computation and machine learning. MIT Press. ISBN: 978-0-262-03561-3. URL: http://www.deeplearningbook.org/ (cit. on p. 3).

Hafner, D., T. P. Lillicrap, M. Norouzi, and J. Ba (2021). "Mastering Atari with Discrete World Models". In: *Proceedings of the International Conference on Learning Representations (ICLR'21)*. Published online: iclr.cc. URL: https://openreview.net/forum?id=0oabwyZbOu (cit. on p. 23).

Hallak, A., D. Di Castro, and S. Mannor (2015). "Contextual Markov Decision Processes". In: *arXiv:1502.02259 [stat.ML]* (cit. on p. 12).

Hansen, N. (2006). "The CMA evolution strategy: a comparing review". In: *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*. Ed. by J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea. Springer, pp. 75–102 (cit. on p. 12).

Hansen, N. and A. Ostermeier (2001). "Completely Derandomized Self-Adaptation in Evolution Strategies". In: *Evolutionary Computation* 9, pp. 159–195 (cit. on pp. 12, 13).

Heath, D. and D. Allum (1997). "The Historical Development of Computer Chess and its Impact on Artificial Intelligence". In: *Deep Blue Versus Kasparov: The Significance for Artificial Intelligence, Collected Papers from the 1997 AAAI Workshop*. AAAI Press, p. 63 (cit. on p. 3).

Helmert, M. (2006). "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26, pp. 191–246 (cit. on p. 13).

Henderson, P., R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger (2018). "Deep reinforcement learning that matters". In: *Proceedings of the Thirty-Second Conference on Artificial Intelligence (AAAI'18)*. Ed. by S. McIlraith and K. Weinberger. AAAI Press (cit. on pp. 186, 188).

Hessel, M., J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver (2018). "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *Proceedings of the Thirty-Second Conference on Artificial*

Intelligence (AAAI'18). Ed. by S. McIlraith and K. Weinberger. AAAI Press, pp. 3215–3222 (cit. on p. 24).

Hutter, F., H. Hoos, and K. Leyton-Brown (2011). "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*. Ed. by C. Coello. Vol. 6683. Lecture Notes in Computer Science. Springer, pp. 507–523 (cit. on p. 189).

Hutter, F., H. Hoos, K. Leyton-Brown, and T. Stützle (2009). "ParamILS: An Automatic Algorithm Configuration Framework". In: *Journal of Artificial Intelligence Research* 36, pp. 267–306 (cit. on p. 5).

Hutter, F., L. Kotthoff, and J. Vanschoren, eds. (2019). *Automated Machine Learning: Methods, Systems, Challenges*. Vol. 5. The Springer Series on Challenges in Machine Learning. Available for free at http://automl.org/book. Springer (cit. on p. 19).

Izquierdo, S., J. Guerrero-Viu, S. Hauns, G. Miotto, S. Schrodi, A. Biedenkapp, T. Elsken, D. Deng, M. Lindauer, and F. Hutter (2021). "Bag of Baselines for Multi-objective Joint Neural Architecture Search and Hyperparameter Optimization". In: *Workshop on Automated Machine Learning (AutoML@ICML'21)* (cit. on p. 17).

Kingma, D. and J. Ba (2015). "Adam: A Method for Stochastic Optimization". In: *Proceedings of the International Conference on Learning Representations (ICLR'15)*. Published online: `iclr.cc` (cit. on p. 19).

Kirk, R., A. Zhang, E. Grefenstette, and T. Rocktäschel (2021). "A Survey of Generalisation in Deep Reinforcement Learning". In: *arXiv:2111.09794 [cs.LG]* (cit. on p. 15).

Kulkarni, T. D., K. Narasimhan, A. Saeedi, and J. Tenenbaum (2016). "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation". In: *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16)*. Ed. by D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates (cit. on p. 22).

Kumar, M. Pawan, B. Packer, and D. Koller (2010). "Self-Paced Learning for Latent Variable Models". In: *Proceedings of the 23rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'10)*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Curran Associates, pp. 1189–1197 (cit. on p. 15).

LeCun, Y., P. Haffner, L. Bottou, and Y. Bengio (1999). "Object Recognition with Gradient-Based Learning". In: *Shape, Contour and Grouping in Computer Vision*. Ed. by D. A. Forsyth, J. L. Mundy, V. Di Gesù, and R. Cipolla. Vol. 1681. Lecture Notes in Computer Science. Springer, p. 319. DOI: `10.1007/3-540-46805-6\_19`. URL: `https://doi.org/10.1007/3-540-46805-6%5C_19` (cit. on p. 24).

Levine, S. and P. Abbeel (2014). "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics". In: *Proceedings of the 27th International Conference on Advances in Neural Information Processing Systems (NeurIPS'14)*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger. Curran Associates, pp. 1071–1079 (cit. on p. 12).

Levine, S. and V. Koltun (2013). "Guided Policy Search". In: *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*. Ed. by S. Dasgupta and D. McAllester. Omnipress, pp. 1–9 (cit. on p. 12).

Li, K. and J. Malik (2017). "Learning to Optimize". In: *Proceedings of the International Conference on Learning Representations (ICLR'17)*. Published online: `iclr.cc` (cit. on p. 12).

Lindauer, M., K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter (2022). “SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization”. In: *Journal of Machine Learning Research (JMLR) – MLOSS* 23.54, pp. 1–9 (cit. on pp. 17, 189).

Lindauer, M., K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter (2019a). “BOAH: A Tool Suite for Multi-Fidelity Bayesian Optimization & Analysis of Hyperparameters”. In: *arXiv:1908.06756 [cs.LG]*. URL: https://arxiv.org/abs/1908.06756 (cit. on p. 18).

Lindauer, M., M. Feurer, K. Eggensperger, A. Biedenkapp, and F. Hutter (2019b). “Towards Assessing the Impact of Bayesian Optimization’s Own Hyperparameters”. In: *IJCAI 2019 DSO Workshop*. Ed. by P. De Causmaecker, M. Lombardi, and Y. Zhang (cit. on p. 18).

López-Ibáñez, M., J. Dubois-Lacoste, T. Stützle, and M. Birattari (2011). *The irace package, Iterated Race for Automatic Algorithm Configuration*. Tech. rep. IRIDIA, Université Libre de Bruxelles, Belgium (cit. on p. 189).

Malashin, R. O. (Sept. 6, 2021). “Sparsely Ensembled Convolutional Neural Network Classifiers via Reinforcement Learning”. In: *6th International Conference on Machine Learning Technologies (ICMLT’21)*. ICMLT 2021, pp. 102–110. ISBN: 9781450389402. DOI: 10.1145/3468891.3468906 (cit. on p. 19).

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015). “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540, pp. 529–533 (cit. on pp. 10, 13, 24).

Obando-Ceron, J. S. and P. S. Castro (2021). “Revisiting Rainbow: Promoting more insightful and inclusive deep reinforcement learning research”. In: *Proceedings of the 38th International Conference on Machine Learning (ICML’21)*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 1373–1383 (cit. on p. 24).

OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang (2019). “Solving Rubik’s Cube with a Robot Hand”. In: *arXiv:1910.07113 [cs.LG]* (cit. on p. 22).

Parker-Holder, J., R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, F. Hutter, and M. Lindauer (2022). “Automated Reinforcement Learning (AutoRL): A Survey and Open Problems”. In: *Journal of Artificial Intelligence Research (JAIR)* 74, pp. 517–568 (cit. on pp. 17, 20, 188).

Pettinger, J. and R. Everson (2002). “Controlling genetic algorithms with reinforcement learning”. In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pp. 692–692 (cit. on p. 20).

Rajan, R., J. L. B. Diaz, S. Guttikonda, F. Ferreira, A. Biedenkapp, J. O. von Hartz, and Frank Hutter (2020). “MDP Playground: Controlling Dimensions of Hardness in Reinforcement Learning”. In: *arXiv:1909.07750*. URL: https://arxiv.org/abs/1909.07750 (cit. on p. 18).

Rice, J. (1976). “The Algorithm Selection Problem”. In: *Advances in Computers* 15, pp. 65–118 (cit. on p. 4).

Sakurai, Y., K. Takada, T. Kawabe, and S. Tsuruta (2010). "A Method to Control Parameters of Evolutionary Algorithms by Using Reinforcement Learning". In: *Proceedings of Sixth International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*. Ed. by K. Yétongnon, A. Dipanda, and R. Chbeir. IEEE Computer Society, pp. 74–79 (cit. on p. 20).

Sass, René, Eddie Bergman, André Biedenkapp, Frank Hutter, and Marius Lindauer (2022). "DeepCAVE: An Interactive Analysis Tool for Automated Machine Learning". In: *Workshop on Adaptive Experimental Design and Active Learning in the Real World (ReALML@ICML'22)*. DOI: 10.48550/arXiv.2206.03493 (cit. on p. 18).

Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61, pp. 85–117 (cit. on p. 24).

Shala, G., A. Biedenkapp, N. Awad, S. Adriaensen, M. Lindauer, and F. Hutter (2020). "Learning Step-Size Adaptation in CMA-ES". In: *Proceedings of the Sixteenth International Conference on Parallel Problem Solving from Nature (PPSN'20)*. Ed. by T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, and H. Trautmann. Lecture Notes in Computer Science. Springer, pp. 691–706 (cit. on pp. 16, 87).

Sharma, M., A. Komninos, M. López-Ibáñez, and D. Kazakov (2019). "Deep reinforcement learning based parameter control in differential evolution". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Ed. by M. López-Ibáñez. ACM, pp. 709–717 (cit. on p. 20).

Solomonoff, R. J. (1957). "An inductive inference machine". In: *IRE Convention Record, Section on Information Theory*. Vol. 2. Institute of Radio Engineers New York, pp. 56–62 (cit. on p. 3).

Speck, D., A. Biedenkapp, F. Hutter, R. Mattmüller, and M. Lindauer (2021). "Learning Heuristic Selection with Dynamic Algorithm Configuration". In: *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21)*. Ed. by H. H. Zhuo, Q. Yang, M. Do, R. Goldman, S. Biundo, and M. Katz. AAAI, pp. 597–605 (cit. on pp. 16, 105).

Sutton, R. S. (1988). "Learning to Predict by the Methods of Temporal Differences". In: *Machine Learning* 3, pp. 9–44. DOI: 10.1007/BF00115009. URL: https://doi.org/10.1007/BF00115009 (cit. on p. 24).

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. 2nd ed. Adaptive computation and machine learning. MIT Press (cit. on pp. 3, 21, 23, 24).

Turner, R., D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon (2021). "Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020". In: *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*. Ed. by H. Escalante and K. Hofmann. PMLR, pp. 3–26 (cit. on p. 18).

van Hasselt, H. (2010). "Double Q-learning". In: *Proceedings of the 23rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'10)*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates (cit. on p. 25).

van Hasselt, H., A. Guez, and D. Silver (2016). "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16)*. Ed. by D. Schuurmans and M. Wellman. AAAI Press, pp. 2094–2100 (cit. on pp. 12, 13, 24, 25).

Watkins, C. J. C. H. (1989). “Learning from Delayed Rewards”. PhD thesis. Cambridge, United Kingdom: King’s College (cit. on pp. 12, 24).

Watkins, C. J. C. H. and P. Dayan (1992). “Q-learning”. In: *Machine learning* 8.3, pp. 279–292 (cit. on p. 24).

Xu, C., T. Qin, G. Wang, and T. Liu (2017). “Reinforcement learning for learning rate control”. In: *arXiv:1705.11159 [cs.LG]* (cit. on p. 19).

Xu, L., H. Hoos, and K. Leyton-Brown (2010). “Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection”. In: *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI’10)*. Ed. by M. Fox and D. Poole. AAAI Press, pp. 210–216 (cit. on p. 6).

Xu, Z., A. M. Dai, J. Kemp, and L. Metz (2019). “Learning an Adaptive Learning Rate Schedule”. In: *arXiv:1909.09712 [cs.LG]* (cit. on p. 19).

Zhang, B., R. Rajan, L. Pineda, N. Lambert, A. Biedenkapp, K. Chua, F. Hutter, and R. Calandra (2021). “On the Importance of Hyperparameter Optimization for Model-based Reinforcement Learning”. In: *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Ed. by A. Banerjee and K. Fukumizu. Proceedings of Machine Learning Research (cit. on p. 17).

Dynamic Algorithm Configuration