Diversity in Reinforcement Learning

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Arno Berengar Feiden

aus

Berlin

Bonn, März 2026

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

Gutachter/Betreuer: Prof. Dr. Jochen Garcke
Gutachterin: Prof. Dr. Ira Neitzel

Tag der Promotion: 02.07.2026
Erscheinungsjahr: 2026

Kurzfassung

Vielfalt im verstärkenden Lernen

Verstärkendes Lernen ist ein Paradigma des maschinellen Lernens, das darauf abzielt, optimale Strategien für zeitdiskrete Steuerungsprobleme auf der Grundlage quantitativer Rückmeldungen zum Verhalten des Lernenden zu entwickeln. Verstärkendes Lernen wird als gradientenbasiertes Optimierungsverfahren formuliert, konkrete Probleme haben allerdings selten eindeutige optimale Lösungen. Diese Arbeit konzentriert sich darauf, Strategien zu unterscheiden, die dasselbe Problem unterschiedlich lösen.

Eine gezielte Analyse der Literatur zum verstärkenden Lernen, insbesondere zu Gradientenverfahren auf Strategien, zeigt, dass diese am besten anhand ihres Verhaltens verstanden werden: anhand der Aktionen, die die Strategie auf Grundlage des Systemzustands auswählt. Es gibt zwei gleichwertige, zentrale Ansätze, um dieses Verhalten empirisch zu verstehen: die episodische Verhaltenscharakterisierung, die eine Strategie als Zufallsvariable im Raum der Zustands-Aktions-Zeitreihen interpretiert, und die Verhaltenscharakterisierung durch die Belegungsmaßzahl, die eine Strategie als Zufallsvariable im Raum der Zustands-Aktions-Paare interpretiert.

Beide Charakterisierungen können verwendet werden, um allgemeine, problemunabhängige Distanzmaße abzuleiten, entweder durch den Vergleich von Zeitreihen oder durch den Vergleich von Stichprobenverteilungen. Ihr Nutzen wird zunächst durch die Visualisierung kleiner Populationen von Strategien untersucht. Die Beziehungen zwischen den Strategien, die sich in der Visualisierung zeigen, spiegeln auch andere, verborgene Beziehungen wider, was den Ansatz validiert.

Diese Idee wird dann in einen evolutionären Algorithmus integriert. Evolutionäres Rechnen ist eine Sparte von Optimierungstechniken, die Populationen von potenziellen Lösungen iterativ verbessern. Insbesondere bei Qualitäts-Diversitäts-Methoden werden die Individuen dieser Population dazu angeregt, unterschiedlich zu sein, um den evolutionären Prozess zu verbessern. Wenn solche Methoden zur Lösung von Problemen des verstärkenden Lernens eingesetzt werden, ist ein Distanzmaß zwischen Strategien erforderlich. Die entwickelten Techniken zur Unterscheidung von Strategien lassen sich erfolgreich mit MAP-Elites verbinden, einer Qualitäts-Diversitäts-Heuristik, die häufig für Probleme des verstärkenden Lernens verwendet wird und Strategien in der Regel anhand selektiver, problemspezifischer Kriterien unterscheidet.

Die Anwendungen validieren die hier etablierten, allgemeinen Distanzmaße, zeigen aber auch, dass eine vielfältige Population von Lösungen zum Beispiel explorativer und robuster gegenüber Umweltveränderungen ist, selbst wenn die Vielfalt nicht auf ein bestimmtes Ziel wie Exploration oder Robustheit ausgerichtet ist. Das offenbart den intrinsischen Wert von Vielfalt.

Abstract

Diversity in Reinforcement Learning

Reinforcement learning is a machine learning paradigm that aims to develop optimal policies for discrete-time control problems based on quantitative feedback on the learner's behaviour. Reinforcement learning is formulated as a gradient-based optimisation procedure, but reinforcement learning problems with unique optimal solutions are rare. This work focuses on ways to distinguish policies that solve the same reinforcement learning problem differently.

A pointed analysis of reinforcement learning literature, specifically of policy gradient methods, shows that policies are best understood through their behaviour: The actions that the policy selects based on the encountered state. There are two equivalent main approaches to understanding this behaviour empirically: Episodic behavioural characterisation, which interprets a policy as a random element in the space of state-action time series and behavioural characterisation through the occupancy measure, which interprets a policy as a random element representing the visitation frequency in the space of state-action pairs.

Both characterisations can be used to deduce general, problem-agnostic distance measures either by comparing time series or by comparing sampled distributions. Their utility is first explored by visualising small populations of policies. The relationships between policies that emerge in the visualisation also reflect other, hidden relationships, thereby validating the approach.

This idea is then integrated with an evolutionary algorithm. Evolutionary computation is a branch of optimisation techniques that improve populations of candidate solutions. In quality-diversity methods specifically, the members of the population are encouraged to differ from one another to improve the evolutionary process. When these methods are used to solve reinforcement learning problems, a measure of distance between candidate solutions is required. The developed techniques to distinguish policies are successfully connected with MAP-Elites, a quality-diversity heuristic that is often used for reinforcement learning problems and typically distinguishes policies based on selective, problem-specific criteria.

The applications validate the established general distances measure, but they also show that a diverse population of solutions is, for example, more explorative and more robust against environmental changes even if the diversity is not directed towards the specific goal such as exploration or robustness. This reveals the intrinsic value of diversity.

Contents

Acronyms

Notation (glossaries)	Description (glossaries)
A2C	Advantage Actor-Critic, a reinforcement learning algorithm
ARS	Augmented Random Search, a reinforcement learning algorithm
AURORA	AUtonomous RObots that Realise their Abilities, a quality-diversity algorithm
BD	Behavioural descriptor, a random element that is defined by selecting and processing observed (random) behaviour of a policy into a numerical value
CMA	Covariance matrix adaptation, a component in algorithms that replaces drawing random noise with drawing from $\mathcal{N}(m, C)$ with mean and variance that may change over time
CVT	Centroidal Voronoi tessellation, a technique to partition a metric space
DAgger	Dataset Aggregation, a behavioural cloning algorithm
DNS	Dominated novelty search, a quality-diversity algorithm
DQN	Deep Q-Learning, a reinforcement learning algorithm
DTW	Dynamic time warping, a technique to align two time series that can be used to define a distance between two time series
ES	Evolution strategies, a branch of optimisation that is inspired from evolution
FPS	Farthest point sampling, an algorithm to sample a data set
GORP	Greedy Over Random Policy, a reinforcement learning algorithm
GPU	Graphics processing unit, specialised computing component that excels at parallel calculations
MAP-Elites	Multi-dimensional Archive of Phenotypic Elites, a quality-diversity algorithm
MDS	Multidimensional scaling, a generic data visualisation technique

Notation (glossaries)	Description (glossaries)
ME	Even shorter abbreviation of MAP-Elites
NEAT	NeuroEvolution of Augmenting Topologies, an evolutionary algorithm
OT	Optimal transport and optimal transport distance
PG	Policy gradient, a gradient on the weights of a policy
PGA	Policy gradient assistance, as in PGA-ME for "Policy Gradient Assisted MAP-Elites". An approach to incorporate gradient information into an evolutionary and typically gradient-free optimisation algorithm
PPO	Proximal Policy Optimization, a reinforcement learning algorithm
QD	Quality-diversity, a category of evolutionary optimisation strategies
RAM	Random-access memory, a form of computer memory
RGB	Red, green, blue: Additive colour model to represent the colour value of pixel on computer screens
SAC	Soft Actor-Critic, a reinforcement learning algorithm
Sgnt	Signature, as in signaure transform. If followed by a number that number indicates the depth
TD3	Twin Delayed Deep Deterministic policy gradient, a reinforcement learning algorithm
TRPO	Trust Region Reinforcement Learning, a reinforcement learning algorithm

List of Symbols

Notation (glossaries)	Description (glossaries)
a	Action, element of an action space
A	Action space of a Markov decision process
A_s	State dependent action space of a Markov decision process
A^π	Advantage function of a policy π
D	Starting state distribution of a Markov decision process
d_π	The discounted occupancy measure of a policy π
$\mathrm{Dir}(\alpha)$	The Dirichlet distribution with concentration parameter $\alpha = (\alpha_1, \ldots, \alpha_K)$ with $K > 1$. It samples points in the $(K-1)$-simplex Δ^{K-1}
h	The entropy of data
J	An utility function that is designed to be minimised or maximised in an optimisation procedure
L	A loss function. Target to be minimised in an optimisation procedure
p	Stochastic transition dynamics of a Markov decision process. Can be written as $p(s'\|s,a)$ to indicate the probability of transitioning to state s' given state s and action a and $p(s',r\|s,a)$ to further indicate that transitioning to state s' with reward r
Q^*	Optimal state-action value function of a Markov decision process
Q^π	State-action value function of a policy π
r	Reward in a Markov decision process
R	Deterministic reward function of a Markov decision process
s	State, element of a state space
S	State space of a Markov decision process
s_0	Starting state with which an episode in a Markov decision process is initialised

Notation (glossaries)	Description (glossaries)
S_{term}	Subspace of terminal states
$S^k(X)$	The signature of depth k of a time series X
t	Time step in time series
T	Deterministic transition properties of a Markov decision process
t_{term}	Time horizon in a Markov decision process
V^*	Optimal value function of a Markov decision process
V^π	Value function of a policy π
α	Hyperparameter of an algorithm: The learning rate in a gradient descent algorithm. The temperature in the SAC algorithm. Regularisation term in the Sinkhorn distance. The concentration parameter of the Dirichlet distribution
γ	Discount factor of a Markov decision process
Δ^n	The standard n-Simplex, all points in $\mathbb{R}^{n+1}$ with non-negative entries that sum up to 1
η_D	Function that returns the average performance of a policy, i.e. the average accumulated discounted reward given a distribution of starting states D. The D may be dropped
θ	A vector of numbers that parameterise a function, typically optimised to adhere to a learning task
λ_i	A centroid in a centroidal Voronoi tessellation
ν	A vector of numbers containing white noise, usually sampled from the normal distribution $\mathcal{N}(0,1)$
π	Policy to a Markov decision process. Can be written as $\pi(a\|s)$ to indicate for a stochastic policy the probability that action a is picked by policy π given state s
Π	A space of policies
π^*	Optimal policy to a Markov decision process
π^θ	A function that serves as a policy with a fixed architecture and parameterised by θ
τ	An episode of a Markov decision process and a policy
ϕ	A function that maps an episode to a numerical value according to a behavioural descriptor

Notation (glossaries)	Description (glossaries)
Φ	The space that a behavioural descriptor maps to
$\mathcal{B}$	The Bellman operator
$\mathcal{M}$	A Markov decision process
$\mathcal{N}(\mu,\sigma^2)$	The normal distribution with mean μ and variance σ^2
$\mathcal{R}$	Cumulative discounted reward of an episode
$\mathcal{T}$	The space of episodes in a Markov decision process
$\mathcal{U}_{[a,b]}$	The uniform distribution in the interval $[a,b] \subset \mathbb{R}$

CHAPTER 1

Introduction

Computers play chess [SHS+17], drive cars and control robotic bodies [CCT+15]. They trade stocks, control machines, suggest films and music [LS20], and, recently, they have even started talking to us [GYZ+25; QGH+25]. Machine intelligence is becoming an ever-present feature of modern life, permeating more and more aspects of everyday life and opening new frontiers of thought. Indeed, some people even fantasise about the end of history being brought about by the emergence of a conscious artificial superintelligence [Cha10]. An important factor in increasing the reach of algorithms to new applications is the growing power of machine learning algorithms and big data. Large amounts of data can be created, stored, and processed to infer relationships [DDS+09] and take informed decisions [LS20].

The necessary condition for this is their incredible potential to perform extremely complex and lengthy algorithmic calculations. *Moore's law* [Moo65] is an observation from 1965 that the number of transistors, and therefore average computing power, doubles in an average microchip roughly every two years. Remarkably, this exponential growth in computing power holds until today [RRM23]. Even if physical limits stop the development of miniaturisation in the near future, other technological advances, such as the current incredible parallelisation potential of graphics processing units or future advances in quantum computing, may continue to drive this exponential increase.

This awesome growth in computing power can quickly deprecate software that has been designed and optimised to work for a particular generation of computers, which is unable to fully utilise the increasing capacities of newer computers. This makes it increasingly attractive to develop meta-algorithms that can solve larger classes of problems, rather than prescribing efficient algorithms for specific problems.

Machine learning algorithms offer precisely that. They can be scaled up with growing computing power, with the same class of algorithms covering a vast range of problems. This paradigm shift was most poignantly described in Rich Sutton's *bitter lesson* [Sut19]. Even the most sophisticated algorithms fortified with domain knowledge and problem-specific optimisation strategies will eventually be surpassed by learning algorithms, rendering these domain-specific optimisations obsolete. Therefore, the bitter lesson for machine learning researchers is to search for general, problem-agnostic approaches that scale, even if they are less performant at the moment of development, rather than optimising for specific problems.

This work aims to examine a specific aspect of one branch of machine learning algorithms: Imagine two agents that each take their own decisions to tackle the same type of problem. Is it possible to quantify how similar or how different their strategies are? Is it possible to do that in a general way, independent of how they formulate their strategies and independent of the problem? And, are there ways to show that this quantification is meaningful?

Chapter 2 introduces reinforcement learning, a class of machine learning based on qualitative feedback in time-dependent problem settings [SB18]. Reinforcement learning addresses problems where an agent has to make decisions by evaluating the present state of its environment. The temporal component requires the agent to quasi-plan and consider actions that may seem unfavourable in the present but lead to better outcomes in the future. In some ways, it is also a self-learning approach, as it gathers its own data through interaction with its environment. The solution to a reinforcement learning problem is a *policy* that dictates the decisions the agent takes.

Is there a meaningful way to quantify the similarity or difference between two solutions to the same reinforcement learning problem? That is the question that started this research. A definitive answer to that question could improve the understanding of problems, algorithms, and solutions of reinforcement learning and contribute to the further development of this field of research. To answer that question, the essence of a solution to a reinforcement learning problem must first be established. Secondly, once a method for quantifying differences has been established, it must be validated. The measured distances must be meaningful to be of use. Indicators for such meaning and possible starting points for this endeavour can be found in the nature of reinforcement learning itself.

At its core, reinforcement learning is an optimisation procedure. The optimum that is targeted is the optimal policy for a given problem. The optimal policy always selects the best action in any conceivable given situation [SB18]. An inconvenient truth is that not every problem that practitioners want to address with reinforcement learning is guaranteed to be convex, which means that it is not guaranteed to have one single optimum that can be deterministically found using optimisation techniques [PSS16b].

This means that unexpected things can happen in practical applications. If the feedback from the environment is sporadic or constant across many situations, there may be a large number or even subspaces of different solutions that all solve the given problem equally well, but an optimisation algorithm typically finds only one solution. One way to better understand such spaces is through algorithms that do not return only one solution, but rather several solutions that sample the space of optimal solutions in a way that covers that space [LS08].

Depending on the shape of the feedback, local optima may exist, where small changes to a policy's behaviour always deteriorate the solution, but a completely different behaviour is flat out better. That is problematic for optimisation procedures as they may get stuck in local optima and then never find the global optima. One practical approach to avoid this problem is to remember the locally optimal solution and search for another locally optimal solution different to the one already found [MC15].

Lastly, most reinforcement learning procedures do not search for a proper global optimal policy that always picks the best course of action. Most procedures look for a locally optimal policy that picks the best course of action only in situations it may actually encounter. This is particularly problematic as the set of situations encountered is contingent on the policy's behaviour during learning, which is bound to change [SWD+17].

A closer look at the theory of reinforcement learning in Chapter 3 reveals that the behaviour of policies is the fundamental object to describe them. The best understanding of a reinforcement learning policy is not that of a function that maps situations to actions. Rather, a policy is best defined by the observed behaviour, that is, the situations it actually encounters due to its reactions and the frequency of these encounters. To cover a space of

equally good solutions and to navigate multiple local optima [MC15], but also to improve learning algorithms [SLA+15], steer away from dangerous behaviour [AHT+17], or learn from demonstration [WCA+19], the best approach is to compare the behaviour of different policies.

Even though the behavioural characterisation of policies is central, it is rarely specifically examined. By collecting the different ways policies are described by their behaviour in theory and practice, a significant research gap takes shape: Behavioural characterisation is either used in an abstract sense [AHT+17; SLA+15] or in a concrete, context-aware sense [CCT+15; PSS16b], with very little middle ground.

This work seeks to address this gap, identifying general, problem-agnostic approaches in Chapter 4 to describe policies through their behaviour that can be utilised for concrete applications that need to differentiate policies. A first and immediate application is the visualisation of cohorts of different solutions. This successful application validates the approaches to characterise policies by finding patterns from the observed behaviour that are not directly encoded in the behaviour, such as which learning algorithm has developed a given policy, or which solution is more robust to environmental changes.

This first half takes an abstract behavioural understanding of policies and moves it into a more concrete area of research. The second half begins from a very concrete, problem-specific understanding of behavioural difference and seeks to replace some of these problem-specific characterisations with the previously derived general, problem-agnostic approaches. The starting point for that is Chapter 5, introducing evolutionary computing for reinforcement learning.

Evolutionary computing [BKS+23] describes a large area of research that is concerned with optimisation procedures inspired by evolution. They utilise populations of solutions that are varied to create new solutions without necessarily discarding earlier ones. In this research area, algorithms have been developed that specifically target reinforcement learning problems. While these algorithms are not competitive with standard reinforcement learning procedures when applied to standard reinforcement learning problems, there are edge cases where evolutionary algorithms bypass typical issues that reinforcement learning algorithms have. One relevant issue that evolutionary algorithms successfully address is exploration.

While the non-convexity of problems can stump reinforcement learning algorithms, certain classes of evolutionary algorithms excel in just that. They are built to keep exploring new solutions regardless of having found a presumed optimum. An especially important branch in that regard is algorithms driven by the concept of novelty [LS08] or diversity [PSS16b]. In the context of reinforcement learning, that means searching for solutions with novel behaviour or fostering a population as a basis for continued learning that exhibits diverse behaviour. Novel behaviour is behaviour that is different from most previously observed behaviour. A population with diverse behaviour means that the behaviour of the different members of the population is different. So, an important mechanism for this class of algorithm is to define what it means for behaviour to be different and how to quantify that.

In reinforcement learning, most problems are shaped in a way that exploration is important, but at the same time, the space of possible solutions is usually so large in comparison to the space of good solutions that indiscriminate search for novelty or diversity results in an excessive search effort. Quality-diversity algorithms address this specific problem by focusing on working with a diverse population whose members also display an appropriate level of fitness towards the presented task [PSS16b].

In Chapter 6, the general behavioural distance measures for reinforcement learning policies are integrated with a quality-diversity algorithm, where they replace problem-specific measures that are typically custom-built for the given reinforcement learning problem and the underlying challenges that the problem statement entails. It addresses the technical difficulties of this integration and concludes by demonstrating on benchmark problems the utility of the generic, data-driven behavioural distance measures derived earlier.

Overall, this work establishes characterisation through their behaviour as the natural way to look at reinforcement learning policies. Established instruments that work with the resulting data are introduced to define several generic behavioural distance measures. These new measures are then validated directly through visualisations that uncover hidden relationships in the data and through application as a component of quality-diversity algorithms, which require proper distance measures between policies to function.

This work shows that reinforcement learning policies can be recognised from their behaviour, that reinforcement learning algorithms create similar policies when restarted, and that similar solutions react similarly to changes in the environment. Measuring the behavioural change gives a more convincing indicator when a reinforcement learning algorithm has converged than the typical tracking of the loss function. It shows that general, data-driven approaches to describe policies by their behaviour are equal to, if not superior to, problem-specific ways to discriminate them when selecting for diversity, which implicitly also demonstrates that diversity, even if not directed towards a certain goal, is still useful, and that there is an intrinsic value to diversity.

Scientific Publications and Contributions

Inevitably, this work includes contributions that have been published before. This includes the content of these invaluable collaborations:

1. **Feiden, Arno** and Jochen Garcke: 'Overcoming Deceptive Rewards with Quality-Diversity'. *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. GECCO Companion 2023: pp. 279–282. doi: 10.1145/3583133.3590741.
2. **Feiden, Arno** and Jochen Garcke: 'Diversity in Reinforcement Learning Through the Occupancy Measure'. *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO 2025: pp. 1235–1245. doi: 10.1145/3712256.3726337.
3. **Feiden, Arno**, Biagio Paparella and Jochen Garcke: 'Generalizing Diversity with the Signature Transform'. *Proceedings of the Genetic and Evolutionary Computation Conference Companion* 2024: pp. 275–278. doi: 10.1145/3638530.3654295.

The chronologically first publication [FG23] reflects on hard exploration environments, MAP-Elites' ability to solve them, and introduces the SquareMaze environment, which is referenced in Subsection 6.3.5. The second publication [FPG24] introduces the idea of robustness, defined here in Subsection 4.4.3, and using the signature transform as a behavioural descriptor, defined here in Subsection 4.1.4. The study on selecting a robust subpopulation in Subsection 6.4.2 also came into being under this collaboration, but was not published. The third publication presented the integration of the occupancy measure in the MAP-Elites framework as a behavioural descriptor, a less developed version of Sections 6.1, 6.2, and 6.3.

CHAPTER 2

Reinforcement Learning

Around the turn of the 20th century, scientists started to examine the mind as a tractable subject. With the advent of psychology and psychoanalysis also came theories of learning that could be validated by experiments on animals. Pavlov's dog, one of the most famous experiments in science history, strikingly demonstrates one core idea: That behaviour can be programmed into a living thing and thus be predicted from previous experiences. It replaces a mystical and spirited view of behaviour with an understanding of life as biological machines that follow rules, some more complex than others, and that are moved by dynamics that can be understood with reason and logic.

A strong idea that proves influential until today is *Thorndike's law of effect*: "Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond." [Tho11]. That is the kernel of behaviourism: the idea that behaviour is formed by interaction with the environment and a positive or negative sensory impression then reinforces or discourages future behaviour of a similar kind.

While from a modern perspective it is clear that not all behaviour can be solely explained from this interaction, it is still recognised as an important mechanism, for example, in cognitive behavioural therapy. The simplicity of the concept and of experiment design, and the significance of the relationship between behaviour and reinforcement made that mechanism an early important milestone for psychology.

This simple and powerful theory of learning inspires an equally simple and powerful approach to machine learning. Reinforcement learning mirrors the three steps of learning described by Thorndike: Behaviour is framed as a response to a situation, a feeling of satisfaction or discomfort is connected with that behaviour, and future reactions to a similar situation are modified based on the sensation. That means, an artificial learning agent is set to interact with an environment, it receives negative and positive feedback to guide its behaviour, and through some mechanism, its behaviour is changed according to that feedback.

Reinforcement learning is a learning paradigm that also profits immensely from the recent advances around deep neural networks [MKS+15; SB18], which can be set up to model the complex relationships of interactions with diverse environments. From the initial conceptual construction of reinforcement learning, some minimal properties can be identified that are essential to reinforcement learning. There needs to be an environment in which interaction is

possible, and in which interactions give positive or negative feedback. Two other important distinguishing features are not direct derivations from the ideas of behaviourism:

Reinforcement learning always assumes a temporal component, so instead of learning from singular interactions, a temporal evolution is envisioned in which decisions in the present can influence the state of affairs later. This is not a necessary addition to mirror Thorndike's law. However, in animal studies, "delay conditioning" is observed: Under the temporal decoupling of interaction with the environment and positive or negative sensation, a change in behaviour is still conditioned [SB18]. So, at least, this addition fits the concept. Another distinctive feature of reinforcement learning is the typical assumption of a time-discrete process.

These assumptions lead to the basic framework of the *Markov decision process*. The basic concept, with several small modifications, shapes the theory and algorithms of reinforcement learning [Alt99; SB18; Sze23]. The framework of the Markov decision process is also the point of contact between the theory and real-world applications. A major breaking point for any application of reinforcement learning is appropriately translating the actual problem into the model framework, specifically, modelling the required positive and negative feedback.

Within the reinforcement learning setup, the learning process itself is set up as an optimisation problem. To make that problem tractable, certain simplifications and estimations are made along the way, which may come with new problems. Most devastatingly, not every problem is guaranteed to be convex. In many cases, theoretical purity is sacrificed for applicability to concrete settings.

Reinforcement learning is a very active field and very much driven by applications. Its most impressive applications are in playing games [VBC+19] and game-like settings [MKS+15], actuating robots [HZA+18], and as a vital component in creating large language models [GYZ+25; QGH+25]. Often, advances in general algorithms, problem-specific tweaks, and new and exciting problem statements overlap and intertwine.

Current research in reinforcement learning has surpassed the search for new improvements of base algorithms seeking to map stimulus and response [HZA+18; SWD+17]. Instead, other elements of artificial intelligence are of interest, such as improved planning and data generation through open-ended learning [SHS+17; Sut25]. Several approaches also concern specifically modified problem sets for specific applications, such as finding and recreating complex sequences [EHL+20], or coordinating multiple agents [BFC+20; BBC+19; HPA+24]. At the same time, theoretical work analyses the algorithmic advances that sometimes are only justified by successful applications to benchmarks [HGA+21; MCD+25].

This chapter introduces reinforcement learning by first contextualising it in Section 2.1 and then introducing the basic algorithmic framework, the Markov decision process with several modifications in Section 2.2 with some immediate derivations and conclusions. More practical aspects of reinforcement learning and a current algorithm are then introduced in Section 2.3. Section 2.4 discusses the state of reinforcement learning research and its reliance on benchmark problems to measure progress. The chapter concludes with a brief summary in Section 2.5.

2.1 Classification of Reinforcement Learning

Reinforcement learning is a form of machine learning and optimisation based on specific assumptions. This section locates reinforcement learning within the broader field of machine learning, contrasting its specifics with those of two distinct domains that begin with similar assumptions: bandits and control theory.

2.1.1 Machine Learning

Machine learning is a vague term that describes an important paradigm in the intersection of mathematics and computer science. It encompasses approaches to solve problems of certain classes, not by prescribing an algorithmic solution but rather by providing data about the specific problem and creating and running an algorithm that uses that data to find a solution. This may be preferable because it takes less human effort to build a solution, but it may also be a creative process in the sense that a solution that is created through machine learning may take a different route than an algorithm built by a human expert [SHS+17; Sut19]. A popular way to further break down the concept of machine learning is the division into unsupervised and supervised learning.

In unsupervised learning, there is no feedback offered or used to check and improve on results. The typical task is finding structure in a given data set X. The result of any unsupervised learning algorithm cannot be evaluated in itself, but needs some additional input, either more information on the data or an evaluation of its value for another later task. Typical tasks include clustering of data points or finding an effective dimension reduction for a given data set.

In supervised learning, the feedback is delivered as labels y. Any data point in X is equipped with a label in y, and learning means finding a function $f : X \to y$ that best matches data and labels. Here, any data point in X has a correct value in y associated with it. For example, the data points may be pictures, and the labels describe if a picture contains a certain object [DDS+09]. Any response to a data point that a solution gives can be evaluated as the right or wrong response. Consequently, evaluations of a learned model in supervised learning consist of aggregations of false and true positives and negatives, for example, in *time series classification* [RFL+21].

Belying the dichotomy implied by the names of these two concepts, reinforcement learning fits into neither category. In reinforcement learning, similar to the setup of supervised learning, a function is learned that matches some input data point to some output. This output is then evaluated, but not through a binary response of true or false, but through the gradual feedback of a reward function. In a very generalising view, unsupervised learning uses no feedback, supervised learning uses qualitative feedback, and reinforcement learning uses quantitative feedback. Also, in stark contrast to supervised and unsupervised learning, reinforcement learning algorithms generate their own data to learn from by interacting with the presented problem.

2.1.2 Bandits

Understanding Reinforcement Learning as a form of supervised learning with gradual instead of binary feedback is an intriguingly neat concept, but it is also driven by concrete applications that come with specific constraints and exploits. The simplest setup presupposing the idea of gradual feedback is a *(multi-armed) bandit* problem [LS20].

"One-armed bandit" is a colloquial term for slot machines, gambling machines that leave no choice to the player, other than to stop playing or continue pulling the one arm of the machine. Multi-armed bandits imagine a machine equipped with several arms, which means several options of play, with the assumption that each arm potentially distributes different returns. Learning is necessary, as the system dynamics are unknown, but those dynamics can be observed by playing. While reinforcement learning usually distinguishes a learning phase from an online phase where a model is deployed or tested, for bandits, both learning and

deployment are concurrent. Obviously, training with infinite tries is trivial. Simply testing any possibility a growing number of times will, in the limit, give a perfect understanding of the environment. The challenge for bandits is therefore mostly set by effective "learning on the job".

This base problem, with some modifications, finds a wide variety of applications, including in the backrooms of modern software, such as recommender systems, advanced A/B-testing, and network routing. Depending on the application, the result of playing the bandit may be modified by *context*. Instead of a discrete number of arms, the input may be continuous, or there may be specifications on the desired goal, such as using as little data as possible and balancing exploring new options and maximising return. However, the most basic reinforcement learning setup is usually considered to feature a temporal evolution, so a setting where a series of actions must be executed and these actions change the state of the system. Bandits in turn, can be seen as an episodic special case of this general reinforcement learning setup with just one time step.

Reinforcement learning is a more specific subset of problems than just learning from quantitative feedback. It also specifies learning in a dynamical system that evolves over time, and that needs to be navigated. That complicates the problem further, as successful learning also requires navigating the temporal evolution of the dynamic system.

2.1.3 Control Theory

Taking decisions in a dynamical system over time is also the subject of *control theory* [LVS12]. The starting point for control problems is a dynamical system that evolves depending on both a state and a control parameter. Control theory typically assumes a continuous-time evolution of the system, while reinforcement learning progresses the system in discrete time steps. The fundamental task consists of finding a controller, which is a function that maps states to controls in a way that a predefined goal is fulfilled. This goal is typically defined as minimising a cost function derived from the dynamics of the system.

The fundamental approach to control theory is to develop an understanding of the dynamical system and derive a proper control strategy from there. From the model of the dynamical system, a controller can be directly derived. This process is substantiated with formal guarantees about stability, robustness, and performance. Control theory also, to a certain degree, follows a learning paradigm. This is evident when building a model from observations to then create a controller based on this model, or when parameters are adjusted online to correct in adaptive control.

2.2 Markov Decision Process

The *Markov decision process* [Put94] constitutes the basic framework of reinforcement learning. In general terms, a Markov decision process is a dynamical system with discrete time steps that evolves based on its state and on actions taken by a controller. Only the last state of the process and the last action taken influence the evolution of the state. That is the *Markov property*. Finally, there is a quantitative feedback on the value of the state and action the process is in at any given time step. Instead of the typical cost function in control theory that is to be minimised, reinforcement learning traditionally works with a reward function that is to be maximised.

More rigorously described, the reinforcement learning problem is an optimisation problem stated as finding a reward-maximising controller in a Markov decision process. First, the

Markov decision process and several important modifications will be described here. Then, the optimisation target and with it the concept of value functions come into play to finally show that, given a proper problem setup, the reinforcement learning problem is well posed with the existence of an optimal policy. The most basic instance of this problem class is a deterministic Markov decision process:

Definition 1 (Deterministic Markov decision process)**.** *A deterministic Markov decision process is a tuple* (S, A, T, R) *of a finite set of* states $s \in S$ *and* actions $a \in A$*, a* transition function $T : S \times A \to S$*, and a bounded* reward *function* $R : S \times A \to \mathbb{R}$*. The product space* $S \times A$ *is called the* state-action space*.*

The concept of the Markov decision process constitutes the starting point for the optimisation problem. Given a combination of state and action, the Markov decision process only returns a new state and the reward of that state-action pair. The next action has to be picked by some other entity. A structured way to select the next action based on the current state is a policy:

Definition 2 (Policy)**.** *A* deterministic policy *on a deterministic Markov decision process* (S, A, T, R) *is a function* $\pi : S \to A$*.*

The Markov decision process is the *environment* an *agent* operates in following a policy. Combining the Markov decision process with a policy results in an alternating process once started. Given a state, the policy returns an action. Given the state and action, the transition function of the Markov decision process returns the next state and the associated reward, see Figure 2.1. Given a Markov decision process, a starting point, and a policy, a time series can be created by this interaction.

Definition 3 (Episode)**.** *Let* $s_0 \in S$*, let* (S, A, T, R) *be a deterministic Markov decision process, and* π *be a policy. The series* τ *with elements in* $(S \times A)$ *defined as* $(s_t, a_t)_{t \in \mathbb{N}_0}$*, where* $s_{t+1} = T(s_t, a_t)$*,* $a_t = \pi(s_t)$ *for* $t \in \mathbb{N}_0$ *is called an* episode *or a* rollout *of* π *given* s_0*. Creating such an episode is also called* rolling out *the policy.*

Let $\mathcal{T} = (S, A)^{\mathbb{N}}$ *be the space of episodes. The reward* $r_t = R(s_t, a_t)$ *is a derived property that can also be understood as part of the episode.*

From this arises the question of how to find a good policy for a given Markov decision process. A good policy means collecting high rewards over the course of an episode. Better yet, to search for a policy π^* that even maximises the accumulated reward over time of such an episode $\sum_t r_t$ given any initial state s_0. However, the properties of the deterministic Markov decision process from Definition 1 do not require that sum to be bounded, so a common addition to the problem setup is a *discount factor* $\gamma \in (0, 1)$, that is added to the definition of the Markov decision process:

Definition 4 (Discounted Markov decision process and cumulative discounted reward)**.** *Let* (S, A, T, R) *be a deterministic Markov decision process. For* $\gamma \in (0, 1)$*, the tuple* (S, A, T, γ, R) *is a* discounted Markov decision process*.*

Let $\tau_{\pi, s_0} = (s_t, a_t)_{t \in \mathbb{N}_0}$ *be a rollout given a policy* π *and state* $s_0 \in S$*. The sum* $\mathcal{R}(s_0, \pi) = \sum_{t \in \mathbb{N}_0} \gamma^t r_t$ *of the rewards* $r_t = R(s_t, a_t)$ *is the* cumulative discounted reward *given* π *and* s_0*.*

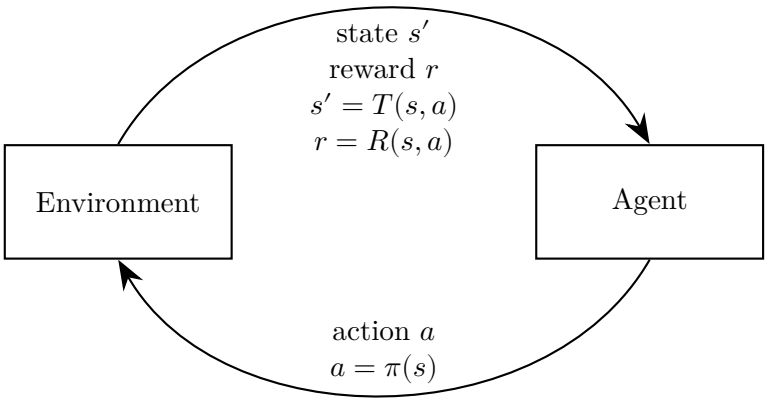


Figure 2.1: Sketch of the Markov decision process loop. Given a state, the agent selects an action. Based on this state and action, the environment then returns a reward and a new state. Here, it is omitted that the definition of a Markov decision process may also include ways to start and terminate the process.

With the assumption that the reward function R is bounded, the optimisation target, the cumulative discounted reward, is now bounded. It is the geometric series $\sum_t \gamma^t r_t \leq \frac{\max(R)}{1-\gamma}$ and therefore the search for an optimal policy is now well posed, even if existence is still left to demonstrate.

Definition 5 (Optimal policy)**.** *Let (S, A, T, γ, R) be a deterministic Markov decision process. The deterministic policy π^* is called an* optimal policy *if for all $s \in S$, the policy π^* maximises the cumulative discounted reward:*

$$\mathcal{R}(s, \pi^*) \geq \mathcal{R}(s, \pi) \qquad \forall \pi : S \to A,\ \forall s \in S \tag{2.1}$$

This framework constitutes the basis of reinforcement learning. Real-world problems that can be posed as a Markov decision process can be addressed with algorithms that search for optimal policies. Several modifications and generalisations can be added to the framework to tackle a wider range of problems.

2.2.1 Extending the Framework

The deterministic Markov decision process, according to Definition 1 is the most basic instance of a Markov decision process. There are countless possible modifications to engage with more diverse problems, to explore a wider solution space, or to narrow down a problem to simplify solving it. The most relevant modifications for the scope of this work are listed here.

Starting and ending an episode

For practical reasons, especially in simulations, reinforcement learning problems usually include conditions under which an episode starts and ends. Also, certain starting configurations may be inherent to a problem. For example, most games start in a certain setup. For the original optimisation problem, to find an optimal policy for any state according to Definition 5, this is not required. Still, some algorithms only look for a locally optimal

solution, a solution that is only optimal when beginning an episode with certain *starting states* (see later in Section 3.1).

Definition 6 (Starting states)**.** *A Markov decision process with* starting states $\mathcal{M} = (S, A, p, D, R)$ *is a tuple with state space S, action space A, transition dynamics p, and a starting state distribution D on S.*

When sampling an episode in $\mathcal{M}$, the starting state $s_0 \sim D$ is sampled from the starting state distribution.

Two important concepts define finality conditions for Markov decision processes: a) *terminal states*, where a episode ends, which represents a "game over" condition that can both signal a successful or unsuccessful conclusion of the problem and b) a *time horizon*, such that no episode is longer than a certain amount of steps in the environment.

Definition 7 (Episodic and continuous Markov decision process)**.** *Let $\mathcal{M} = (S, A, T, \gamma, R)$ be a deterministic Markov decision process.*

Terminal states *in $\mathcal{M}$ are elements of a subset of the state space $S_{term} \subset S$, such that an episode ends when reaching $s \in S_{term}$.*

A time horizon *is a number $t_{term} \in \mathbb{N}$ for which any episode in $\mathcal{M}$ ends at the latest after t_{term} steps.*

If no episode in $\mathcal{M}$ can go on indefinitely, the process is episodic*. Otherwise, it is* continuous*.*

An *episodic* problem removes the conceptual necessity for a discount factor because the cumulative reward is trivially bounded as long as the reward function is bounded. However, a discount factor represents an encouragement to realise rewards sooner rather than later. Including a discount factor, therefore, seems to be a good practice for more robust learning algorithms, even though it is not strictly necessary.

A computational learning algorithm must always transform any given problem into an episodic problem to guarantee an end of calculation. Depending on the learning algorithm, a mock-up of a *continuous* problem, that theoretically could have episodes with infinite timesteps, is still possible. It can be achieved by distinguishing the end of an episode due to a terminal condition or due to reaching the time horizon and adjusting the learning algorithm accordingly (see, for example, Subsection 2.3.3).

Vice versa, an episodic problem can be restated as a continuous problem by adding a virtual ending state. Then, instead of actually ending the episode, that virtual ending state is reached. The ending state cannot be exited with any action and no rewards are given out. Functionally, the episode is over when the ending state is reached. This way, any theoretical reflection on continuous problems can be carried over to episodic problems.

State-dependent actions

Another modification is the inclusion of state-dependent actions. The actions available to an agent may depend on the state of the environment.

Definition 8 (State-dependent actions)**.** *For any state $s \in S$ let A_s be a set of actions. Let $A = \{A_s\}_{s \in S}$ be the collection of all sets of actions. The Markov decision process (S, A, T, γ, R) is said to have* state-dependent actions*.*

A deterministic policy for such a Markov decision process is a function $\pi : S \to \bigcup_{s \in S} A_s$ such that $\pi(s) \in A_s$, $\forall s \in S$.

While this may seem to further complicate a problem, in a practical application, state-dependent actions may actually simplify the problem and with it the learning process. Instead of an enlargement of the action space, it can also be interpreted as a state-dependent reduction of the action space and, with that, a simplification of the problem.

States and observations

Another concession to practical concerns is reflected in the distinction between the *state* and the *observed state* or *observation*. While any of these terms are sometimes used interchangeably, it is implied that even though the observed process may not adhere to the Markov property, underlying it is a hidden Markov process. So, when both terms are used together, they emphasise this distinction between the real, hidden state of the underlying process and the potentially incomplete observation of that state. Similarly, decision processes that may not fully adhere to the Markov property are tackled with methods of reinforcement learning with the tacit assumption that this too is the observable hull of a proper hidden Markov process. This is particularly true in a real-world problem, as with a robotics application. When applied to reality, the Markov property is debatable. Anyway, the full state of the observable universe cannot be processed, so an observation must take the place of a proper state. The observation should be constructed in a way that the real process can reasonably be estimated as a Markov process.

One example of that is frame-stacking, a trick where the state the policy processes is created from a stack of the last frames of the actual observations [MKS+15]. This allows the learner to see at any given point not just the state of the environment but also how the dynamics unfold without giving up the Markov property, see also Subsection 2.3.3.

Stochastic environments and agents

The most important generalisation is the inclusion of randomness into the problem. Both the process and the policy may be stochastic. A stochastic process replaces the transition function with a probability distribution over the space of "next states" for each state-action pair. A stochastic policy is a set of probability distributions over the action space for each state:

Definition 9 (Stochastic Markov decision process and policy)**.** *Let S be a finite state space, A be a finite action space and $R : S \times A \times S \to \mathbb{R}$ be a reward function. For each state-action pair $(s, a) \in S \times A$, let $p_{(s,a)}$ be a probability distribution over S. For $p = \{p_{(s,a)}\}_{(s,a)\in S\times A}$ the tuple (S, A, p, γ, R) is a* stochastic Markov decision process*.*

For a Markov decision process $\mathcal{M}$ with finite state space S and action space A, a collection of probability distributions $\pi = \{\pi_s\}_{s\in S}$ over A is a stochastic policy*.*

In a stochastic Markov decision process, the reward function may also depend on the next state $r_t = R(s_t, a_t, s_{t+1})$. An intuitive way to describe these entities is given by [SB18], writing a stochastic policy as $\pi_s(a) = \pi(a|s)$ and the stochastic transition dynamic as $p(s', r|s, a)$, which already includes the reward dynamic as well.

Generalised Markov decision process

The framework of the Markov decision process is thus extended to a more complex representation:

Definition 10 (Discounted probabilistic Markov decision process with starting states)**.** *A discounted probabilistic Markov decision process with starting states is a tuple* (S, A, p, γ, R, D) *of* states $s \in S$, actions $a \in A_s$ *with* $A_s \in A$, transition probabilities $\{p_{(s,a)}\}_{(s,a)\in S\times A}$ *in* $S \times \mathbb{R}$ *for each state* $s \in S$ *and action* $a \in A_s$*, a discount factor* $\gamma \in (0, 1)$ *and a distribution of starting states* D *in* S*.*

Especially if such a Markov decision process is still finite in state and action space, this is a very tractable object that allows for a broad and flexible view on the subject. The key to work with the probabilistic aspects is to define random elements S_i, A_i that for any policy define how likely it is to encounter a certain state and a certain action in the i-th time step.

Theorem 1 (Probability measure of episodes ([Sze23]))**.** *For a finite Markov decision process* $\mathcal{M} = (S, A, p, \gamma, D)$ *with state space* S*, action space* A*, and transition probability* p*, there exists a measurable space* $(\Omega, \mathcal{F})$ *and a sequence of random elements* $(S_i, A_i)_{i\in\mathbb{N}_0}$ *such that for any policy* π *and any probability measure* D *on* S *that defines initial states, there exists a probability measure* $\mathbb{P}$ *over* $(\Omega, \mathcal{F})$*, such that*

1. $\mathbb{P}(S_0 = s) = D(s)$ $\forall s_0 \in S$ (2.2)
2. $\mathbb{P}(A_t = a|S_t = s) = \pi(a|s)$ $\forall s \in S, a \in A$ (2.3)
3. $\mathbb{P}\left(S_{t+1} = s'|S_t = s, A_t = a)\right) = p(s'|s, a)$ $\forall s, s' \in S, a \in A$ (2.4)

If $(\tilde{\Omega}, \tilde{\mathcal{F}})$, $(\tilde{S}_i, \tilde{A}_i)_{i\in\mathbb{N}_0}$, $\tilde{\mathbb{P}}$ *also fulfil these properties for* π *and* D*, then the joint distribution of* $(S_i, A_i)_{i\in\mathbb{N}_0}$ *under* $\mathbb{P}$ *and of* $(\tilde{S}_i, \tilde{A}_i)_{i\in\mathbb{N}_0}$ *under* $\tilde{\mathbb{P}}$ *are identical.*

Proof. The theorem follows from the Ionescu-Tulcea theorem (Theorem 3.3 in [LS20]) by identifying the tuple $(\mathcal{M}, \pi)$ as a Markov chain in the state space that starts in D and progresses with the Markov kernel $p \circ \pi$ from s_i to s_{i+1}. □

It makes sense to also write explicitly $R_i = R(S_i, A_i, S_{i+1})$ the random element R_i for the reward function. With the Markov decision process and the optimisation target, the cumulative discounted reward $\mathcal{R}$, defined, some fundamental properties can be derived, specifically, the existence of an optimal policy. The important building block for that is the concept of the *value function* and deriving its recursive nature.

2.2.2 Value Functions

A fundamental idea of reinforcement learning is, given a policy, the introduction of the *value* of a state, or of a state-action pair. This quantity captures the realisation of the optimisation target given a certain policy and a certain starting state (and possibly action):

Definition 11 (Value function, Q-function)**.** *Given a Markov decision process* (S, A, p, γ, R) *and a policy* π*, define the* value function of π *on the state space as*

$$V^\pi : S \to \mathbb{R} \tag{2.5}$$

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s_0\right] \tag{2.6}$$

and the state-action value function of π*, sometimes just* Q-function*, as*

$$Q^\pi : S \times A \to \mathbb{R} \tag{2.7}$$

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s_0, A_0 = a_0\right]. \tag{2.8}$$

The value function of a policy π in a state s or in a state-action pair (s,a) is precisely the expected value of the cumulative discounted reward of that policy given either the state s or the state-action pair (s,a) as a starting point of the episode. Both value functions can also be written directly in terms of the cumulative discounted reward $\mathcal{R}$:

$$V^\pi(s) = \mathbb{E}\left[\mathcal{R}(s,\pi)\right] \tag{2.9}$$

$$Q^\pi(s,a) = \mathbb{E}\left[R(s,a,s') + \gamma\mathcal{R}(s',\pi) \mid s' \sim p_{(s,a)}\right] \tag{2.10}$$

Without loss of generality, the factor $(1-\gamma)$ may be used to scale these expressions to be bounded independent from γ by $\max(R)$ or the factor $\frac{1-\gamma}{\max(R)}$ may be used to scale them to be bounded by 1. Similarly, for a Markov decision process with a time horizon t_{term}, the expressions may be scaled by dividing by t_{term} to have the value independent from the time horizon.

A derived property of the value function is the advantage function of a policy. Given a state-action pair (s,a), it describes how much better or worse an action picked by a certain policy given state s is, compared to the action a.

Definition 12 (Advantage function)**.** *Given a Markov decision process* (S,A,p,γ) *and a policy* π*, define the* advantage function $A^\pi : S \times A \to \mathbb{R}$ *of* π *as*

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{2.11}$$

If $a = \pi(s)$, then $A^\pi(s,a) = 0$, but otherwise in any state s, its value is exactly the difference in the value of taking action a over $\pi(s)$.

2.2.3 Existence of an Optimal Policy

The Markov property unlocks a useful recursive formulation of the value function and Q-function in the *Bellman equation*:

Theorem 2 (Bellman equation)**.** *Let* $\mathcal{M} = (S,A,p,\gamma,R)$ *be a Markov decision process and* π *be a policy for it. Let* V^π *be its value function and* Q^π *its action-value function. Then* V^π *and* Q^π *can be written recursively as*

$$V^\pi(s) = \mathbb{E}\left[R(s,a,s') + \gamma V^\pi(s') \mid a \sim \pi(s), s' \sim p_{(s,a)}\right] \tag{2.12}$$

$$Q^\pi(s,a) = \mathbb{E}\left[R(s,a,s') + \gamma Q^\pi(s',a') \mid s' \sim p_{(s,a)}, a' \sim \pi(s')\right]. \tag{2.13}$$

Proof. Let $s_0 \in S$ and π be a stochastic policy for $\mathcal{M}$. Following Theorem 1 with the starting state fixed as s_0 (or the starting distribution D set as the Dirac delta function δ_{s_0}), let

$(R_i)_{i\in\mathbb{N}}$ be the random elements of the reward in the i-th step, and let S_1 be the random element of the state in the first time step. Then

$$\mathcal{R}(s_0,\pi) = \sum_{t\in\mathbb{N}_0} \gamma^t R_t = R_0 + \gamma \sum_{t\in\mathbb{N}_0} \gamma^t R_{t+1} = R_0 + \gamma\mathcal{R}(S_1,\pi) \tag{2.14}$$

and accordingly

$$V^\pi(s_0) = \mathbb{E}\left[\mathcal{R}(s_0,\pi)\right] = \mathbb{E}\left[R_0 + \gamma\mathcal{R}(S_1,\pi)\right]. \tag{2.15}$$

Set $s = s_0$, $s' = s_1$, and $a = a_0$ and note that $R(s_0,a_0,s_1) = R_0$, this resolves to

$$V^\pi(s) = \mathbb{E}\left[R(s,a,s') + \gamma V^\pi(s') \mid a\sim\pi(s), s'\sim p_{(s,a)}\right]. \tag{2.16}$$

For the Q-function, fix again the starting state s_0 but also the first action a_0 and again define the random elements following Theorem 1. The same expansion as in Equation (2.14), but with an incremented index is used:

$$\mathcal{R}(S_1,\pi) = R_1 + \gamma\mathcal{R}(S_2,\pi) \tag{2.17}$$

The first state-action pair $(s_0,a_0) = (s,a)$ is fixed and Q resolves to

$$\begin{aligned} Q(s_0,a_0) &= \mathbb{E}\left[R_0 + \gamma\mathcal{R}(S_1,\pi)\right] && (2.18)\\ &= \mathbb{E}\left[R_0 + \gamma(R_1 + \gamma\mathcal{R}(S_2,\pi))\right] && (2.19)\\ &= \mathbb{E}\left[R(s_0,a_0,s_1) + \gamma Q^\pi(s_1,a_1) \mid s_1\sim p_{(s_0,a_0)}, a_1\sim\pi(s_1)\right]. && (2.20) \end{aligned}$$

Rewriting $s = s_0$, $s' = s_1$, $a = a_0$, and $a' = a_1$ proves the statement. □

This recursive formulation is a key property of the state and state-action value. It detaches the optimality of a policy from the sum over rewards of an episode and instead gives a representation that is based on a transition of a state-action pair (s,a) to the next state s' and reward r. This makes the value much more controllable and makes it possible to show the existence of an optimal policy.

Theorem 3 (Existence of an optimal policy)**.** *Given a Markov decision process (S,A,p,γ,R) with a finite state and action space, there exists an optimal policy π^* in the sense that its value function $V^* := V^{\pi^*}$ is maximised,*

$$V^*(s) \geq V^\pi \tag{2.21}$$

for all states $s\in S$ and for all deterministic policies $\pi : S\to A$ defined for the Markov decision process. The same holds for its action-value function $Q^* := Q^{\pi^*}$ with

$$Q^*(s,a) \geq Q^\pi(s,a) \tag{2.22}$$

for all $s\in S$, $a\in A$ and all policies $\pi : S\to A$.

Proof. Let Π be the space of all deterministic policies which map $S \to A$. Set $V^*(s) := \sup_{\pi \in \Pi} V^\pi(s)$. Then V^* fulfils the *Bellman optimality equation*

$$V^*(s) = \sup_{\pi \in \Pi} V^\pi(s) \tag{2.23}$$

$$= \sup_{\pi \in \Pi} \mathbb{E}\left[R(s,a,s') + \gamma V^\pi(s') \mid a \sim \pi(s), s' \sim p_{(s,a)}\right] \tag{2.24}$$

$$= \max_{a \in A} \mathbb{E}_{s' \sim p(s,a)}\left[R(s,a,s') + V^*(s')\right] \tag{2.25}$$

The maximum can be picked here, because A is finite. The maximum is not necessarily unique. If that is the case, any maximising action can be picked.

A policy π^* that chooses a maximising action in every state is an optimal policy. The analogue is possible for $Q^*(s,a) := \max_{\pi \in \Pi} Q^\pi(s,a)$. With this definition, the value function of π^* is V^*, and the state-action value function of π^* is Q^*. □

The proof also shows that different optimal policies exist if, under certain states, the maximising action is not unique. In particular, any stochastic policy π^* is optimal if for all states $s \in S$ the random element $\pi^*(s)$ of that stochastic policy is any probability distribution where only the actions $a \in A_s$ that maximise $Q^*(s,a)$ or that maximise the expression $\mathbb{E}\left[R(s,a,s') + V^*(s') \mid s' \sim p(s,a), a \sim \pi(s)\right]$ have non-zero probabilities.

From the Bellman equation, the recursive formulation of the value function, the existence of an optimal policy is shown. It shows that the best action can be chosen by selecting only for the immediate reward and the value of the next state. Neither the past nor anything but the nearest future need to be considered besides the value function. All relevant aspects of all future behaviour can be fully encoded in the value function. This breakdown of the optimality of the sequential problem into mere state-dependent optimal decisions is called *Bellman's principle of optimality.*

In Theorem 3, it is noteworthy that the statement was restricted to Markov decision processes with a finite state and action space. The existence of a deterministic optimal policy can also be shown under the more general prerequisite of optimising for a Borel state Markov decision process, for example, under the following conditions [Fei11]:

Definition 13 (Borel state Markov decision process)**.** *The tuple $\mathcal{M} = (S, A, p, R)$ is a* Borel state Markov decision process *if the state space S is a Borel space, the action space A is a complete separable metric space, the transition probability $p(S'|s,a)$ is a probability measure on the Borel σ-field of S, it satisfies the condition that for any Borel subset S' of S the function $p(S'|s,a)$ is measurable in $(s,a) \in S \times A$, and it is setwise continuous in $a \in A$, which means that $p(S'|s,a_n) \to p(S'|s,a)$ for every $s \in S$, every sequence $a_n \to a$, and every measurable subset $S' \subset S$.*

The restrictions in the definition are mostly concerned with setting the Markov decision process up in a way that everything is properly measurable and well-behaved. The main restriction is set on the transition probabilities with the setwise continuity. It is easy to see, why: If the behaviour of a sequence $a_n \to a$ changes only in the limit or if the limit is not a valid action anymore, the equivalent of the switch from supremum to maximum from Equation (2.25) is not possible anymore and particular instances of a Markov decision process can be constructed such that an optimal policy does not exist for them. For the

remainder of this work, mostly for simplicity of notion, finite state and action spaces are the default setting to examine the methods and ideas of reinforcement learning theory.

2.3 The Reinforcement Learning Problem

The optimisation problem that drives reinforcement learning is the search for an optimal policy π^* for any given Markov decision process $\mathcal{M}$. The resolution of this optimisation problem is the *reinforcement learning problem*. There are two main constructive approaches to the search for such an optimal policy: One is searching for an optimal value function and deriving a policy, the other is a direct search in the function space. For either approach, practical concerns complicate the reinforcement learning problem.

Using the Bellman equation as shown in the proof of Theorem 3, an optimal policy exists and can be directly derived from an optimal action-value function: An optimal policy π^* maps any state $s \in S$ to an action in A_s that maximises $Q^*(s, a)$. This makes the development of algorithms possible that attempt to find the optimal action-value function and derive the policy from it.

Searching for an optimal value function and deriving a policy from it is also referred to as a *critic* or *critic-only* method, whereas searching for an optimal policy directly is an *actor* or *actor-only* method, because the policy acts, and the value function is its critic. An approach that does both in a complementary way is an *actor-critic* method. Actor methods will be introduced in Chapter 3.

In practical applications, three common techniques often become part of algorithms: *Function approximation*, *bootstrapping*, and *learning from experience*. Function approximation is necessary to abstract from simple lookup tables, but also to leverage the potential of machine learning methods to generalise past experience to draw conclusions about previously unseen states. Bootstrapping is necessary if a learning algorithm uses the recursive Bellman equation. Due to the recursive formulation, the estimate of the optimal value function is used to update this estimate itself. Learning from experience is a property that has become deeply intertwined with the reinforcement learning problem itself. It means that the learning algorithm creates its own data to learn from by interacting with the environment.

Combining the three practical approaches, function approximation, learning from experience, and bootstrapping to a working algorithm can lead to grave instabilities [SB18]. However, all three are fundamental ideas that are regularly used in modern algorithms, which obliges them to find specific stabilisation techniques that mitigate these problems [HDS+18].

Connected with the idea of learning from experience is *exploration*, an indispensable element of reinforcement learning. The underlying paradigm of reinforcement learning is to learn through interaction with the environment. That means that part of the learning algorithm must be a strategy to interact with the environment in such a way that the experience gained is useful for learning. Specifically, depending on the environment, certain states may only be found after a certain sequence of actions that the agent has to take and these action may not seem to be beneficial without the full knowledge of the environment. So, a learner needs to explore different paths in the environment to successfully learn from experience.

2.3.1 Function Approximation

In a finite Markov decision process, a learning algorithm may search for the actual, concrete values of $Q^*(s, a)$ for every state s and action a in a *Q-table*, a matrix $Q \in \mathbb{R}^{|S| \times |A|}$, where

every entry Q_{ij} encodes the current estimate for the optimal state action value $Q^*(s_i, a_j)$. From this Q-table, an optimal policy can be derived directly. In continuous state or action spaces, either a discretisation of the spaces or a functional representation of the policy becomes necessary.

That means that a policy is defined as a function $\pi : \mathbb{R}^n \times S \to A$, often in a way such that it is differentiable in $\theta \in \mathbb{R}^n$. The definition of π is also called the *architecture* of the function, the values θ are its *parametrisation*. Most reinforcement learning algorithms use a *fixed architecture* of π, which means that only the parametrisation θ changes during learning. In that case, it is convenient to write $\pi^\theta(s) = \pi(\theta, s)$. NEAT [SM02] is an example of an exception where the architecture is also developed over the learning process, which is introduced later in Subsection 5.1.1.

With a fixed architecture, the reinforcement learning problem can then be understood as searching for weights θ^* such that π^{θ^*} is as close as possible to the optimal policy π^* in some norm $d(\cdot, \cdot)$ on the functions.

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^n} d(\pi^*, \pi^\theta) \tag{2.26}$$

This is particularly promising considering the strong ability of neural networks to approximate functions and extrapolate from experience. If the reinforcement learning algorithm learns the weights of a deep neural network, this is sometimes referred to as *deep reinforcement learning*. This paradigm, however, has become so prevalent that deep reinforcement learning is sometimes regarded as the standard approach to reinforcement learning, and techniques that forgo deep neural networks are understood as classic reinforcement learning in contrast.

The formulation of the best function approximation in Equation (2.26), however, is still too vague to work with. There are two important general approaches to function approximation in reinforcement learning. One approach hinges on searching for the optimal value function and deriving a policy from that. An algorithm then exploits the position of the optimal value function as a fixed point of the operator

$$\mathcal{B} : Q(s, a) \to R(s, a, s') + \gamma Q(s', a') \qquad s' = T(s, a),\ a' = \arg\max_{\bar{a} \in A} Q(s', \bar{a}). \tag{2.27}$$

The other important approach is to define a differentiable utility function J that represents the optimisation target. The function is defined with the parameterisation θ for a policy π^θ as its argument in a way that gradient descent on J is possible.

$$\theta' \leftarrow \theta - \alpha \nabla J(\theta) \tag{2.28}$$

Both approaches can only hope to find a global optimum if the problem still maintains the respective relevant properties: For the fixed point approach, that $\mathcal{B}$ is still a proper contraction, for the gradient approach, that the objective function is convex and the step size α is small enough. With sufficient assumptions on the Markov decision process itself and under certain properties of the function approximations, proofs for the convergence of function approximation strategies can be constructed [HGA+21].

2.3.2 Learning from Experience and Bootstrapping

In reinforcement learning, it is assumed that the system's dynamics are unknown, but that interaction with the system is possible. That means, the transition function T can be called, or the transition dynamics p can be sampled from, but the equations that govern these dynamics are inaccessible. Learning is achieved through interaction with the environment. Fundamentally, reinforcement learning means *learning from experience*.

A major improvement that is derived from this idea is *experience replay* [Lin92]. For this idea, the recursive nature of the value function in the Bellman equation is a key element. It enables learning from experience based on transitions, which are tuples containing the data of one step in the environment, so given a certain state s, sampling $a \sim \pi(s)$ and $s' \sim p(s, a)$, and using $r = R(s, a, s')$, creates a transition (s, a, r, s'). This data may be enriched with other relevant information, for example, whether the episode was ended after the transition and whether it was ended by termination or by a time-out. This focus on transitions is motivated by Bellman's principle of optimality, which states that an optimal policy takes the optimising action based on the immediate reward and the value of the next state.

These transitions (s, a, r, s') are stored in a *replay buffer*. From this replay buffer, learning with a stochastic gradient descent method is possible. This approach can greatly improve *sample efficiency*, the rate of learning measured by evaluations of the transition function. However, learning from transitions that were created from a different policy than the one that is being optimised in the current iteration (*off-policy* learning) can lead to problems when the expected experiences of the policies do not fully overlap. Generally, off-policy algorithms need to address this issue, for example, with *importance sampling* [SLA+15; SB18], where the selection of transitions to learn from is weighted by the probability of the current policy to encounter that particular state, see also Subsection 3.2.2.

When learning from transitions, an estimate of the value of the next state $V(s')$ is used when learning the value function V itself. This self-referential setup is referred to as *bootstrapping*.

In practice, learning from experience can also necessitate exploration. The goal is to reduce the problem to only those situations that are relevant for the learner. Not every dynamic needs to be fully understood, but only those that a trained policy actually will encounter. While this reduces the search space and the scope of the problem, it adds another layer of complication, which is to modify the behaviour of the agent in a way that relevant states are found, but still (near-)optimal behaviour is being developed. Subsection 2.3.4 is concerned with exploration.

2.3.3 Q-Learning

Deep Q-Learning (DQN) [MKS+15] marks an important milestone for deep reinforcement learning. It is based on the principle of Q-learning [Wat89], which can be characterised as a critic-only approach. In Q-learning, the action-value function of the optimal policy in a discrete state-action space is estimated from past experiences with updates based on transitions (s, a, r, s') and with a learning rate $\alpha \in (0, 1]$:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a' \in A_s} Q(s', a')) \tag{2.29}$$

The update is aiming to reduce the *temporal difference*

$$r + \gamma \max_{a' \in A_s} Q(s', a') - Q(s, a), \tag{2.30}$$

which should be 0 for the optimal Q-function according to the Bellman equation (2.13). The learning rate, among other things, is necessary to compensate for random fluctuations. The theoretically sound update on the expected value

$$\mathbb{E}_{s' \sim p_{(s,a)}, a' \sim \pi(s')} \left[R(s, a, s') + \gamma Q^{\pi}(s', a') - Q(s, a) \right] \tag{2.31}$$

is only estimated based on one observed transition in Equation (2.29). A small enough learning rate avoids overestimating the importance of that observed transition and makes room for some memory of past experiences.

In DQN, this idea is stabilised to learn the weights of a neural network with a replay buffer and stochastic gradient descent. The technical groundwork for this is found in the advances in fast and stable backpropagation of gradients in neural networks. The iterative updates are taken to minimise the loss function defined as

$$L(\theta) = \left(r + [s' \notin S_{\text{term}}] \, \gamma \max_{a' \in A} Q^{\theta}(s', a') - Q^{\theta}(s, a) \right)^2, \tag{2.32}$$

where $[P]$ is read as 1 if the statement P is true and 0 if it is false. The addition of the term $[s' \notin S_{\text{term}}]$ therefore sets $Q_{|S_{\text{term}}} \equiv 0$ for the learning update, even if the generalisation of the neural network does not comply with that. This loss value provides an example how a continuous problem can be modelled in a simulated environment where an episode may be infinite in theory but not in practice: The end of the episode is only valued as 0, if the episode ended through reaching a terminal state, not by reaching the time horizon.

The impact of DQN can also be explained by the success story it tells, performing well on games of the *Arcade Learning Environment*, a set of emulated Atari 2600 games, which also requires some problem-specific solutions to work, such as frame-stacking, which means taking as the policy's input not just the last state but a stack of the last k states. This idea can be justified by Takens' theorem [Tak81], which establishes a connection between the delay embedding, a time-continuous generalisation of frame-stacking, and the underlying dynamics of the system. Finally, DQN also needs an approach to exploration to manage the vast state space given by any configuration of pixels on a screen.

2.3.4 Exploration

Exploration in reinforcement learning is intimately linked to the paradigm of learning from experience. A machine learning algorithm that can only learn from experience has to gain a wide variety of experiences to navigate the problem properly. The task of collecting a relevant amount of information during learning, even if that information is collected by trying out new, suboptimal ways to interact with the environment, is exploration.

Exploration is a fundamental requirement for reinforcement learning. This is partly due to the complexity of the problems, the intended application to real-world problems, and the uncertainty about the convexity of the optimisation target. Primarily, however, exploration is important and necessary because reinforcement learning problems tend to have a search

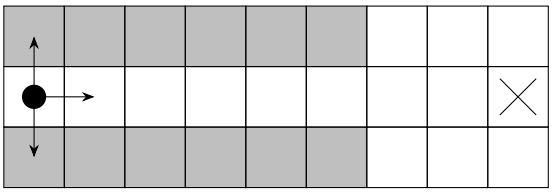


Figure 2.2: Gridworld example for the danger of exploration where the one good solution is reaching a goal state marked with an X after a dangerous corridor that is flanked by greyed out terminal states.

space so large that it cannot be fully examined. Rather, a smart technique to both find good solutions and also explore promising approaches is in order.

The unsolved question of how to best balance these competing objectives is called the *exploration-exploitation dilemma* [SB18]. Iterative optimisation procedures maintain and improve upon an estimate of the best solution at each iteration. In reinforcement learning, at every iteration, there is an estimate for the best policy. Greedily taking the actions this estimate for the best policy suggests, leads to a smaller variety of experiences to learn from. If a policy under a reinforcement learning algorithm converges towards a seemingly optimal policy, that will inevitably narrow the experiences collected to only that seemingly optimal behaviour. However, traversing other parts of the space may lead to even more favourable outcomes. The *overestimation bias* [FHM18], see also Subsection 3.1.5, is one phenomenon that exacerbates this problem.

The problem of exploration is also linked to the restriction on starting states, as it is unclear which states a policy can reach from the starting states and which of these reachable states may lead to favourable outcomes. A possible strategy for exploration is using *exploring starts*, a technique where each episode is started in a completely random state. This ensures that after sufficient learning time, every state in the state space is sampled as a starting state. Due to the Markov property, it does not matter if a state is reached after some time or if an episode started in that state, so every state is being examined in a meaningful way.

However, the restriction to starting states is also a massive simplification that avoids trying to learn strategies for configurations that are imaginable but not physically possible. It is unreasonable to ask an amateur chess player to start learning with Chess960, a variant with a randomised starting position for all pieces but the pawns. While the restriction to starting states simplifies many complex problems, it also creates a new problem. That is, to experience any relevant (high-value) state that a policy can reach from the restricted set of starting states.

Exploration can be understood as an open problem in reinforcement learning. There is no single recognised solution for it, but rather every practical algorithm introduces some strategy to address it. That typically means to gather experiences with a policy that, at any point of the iterative learning process, is not the best candidate, but rather a trade-off between a good candidate and a candidate that continues to explore. For example, in DQN, at any point during the optimisation, an ε-greedy policy is used, which for an $\varepsilon \in (0, 1)$ picks the best option with a probability of $(1 - \varepsilon)$ and a random action otherwise. This policy should perform worse the higher the ε value is, according to the knowledge gathered so far. But it may encounter different states and strategies, and as a consequence, learn faster and better.

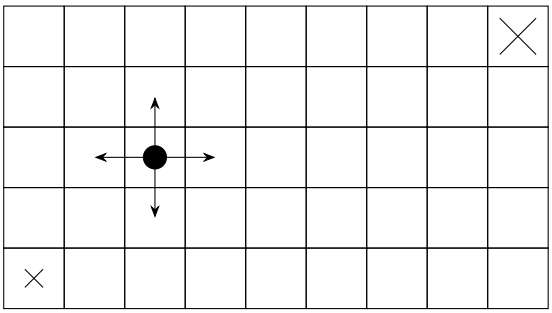


Figure 2.3: Gridworld example for exploration where one, good solution is reaching a goal state in the lower left and one, better solution is reaching a goal state in the upper right corner. It requires exploration to find one solution, but even after that, more exploration to find the second.

Depending on the problem, excessive or unsuitable exploration may not only delay the learning process but even prevent meaningful learning completely. The environment sketched in Figure 2.2 illustrates this danger of exploration. The circle represents the agent, the X marks the goal state, which is terminal and incurs some reward. The shaded cells represent game-ending states that incur no reward. Now, exploration may drive the agent to fall into game-ending states, making it very unlikely to find its way out of the corridor. With a ε-greedy strategy and a very long corridor, it becomes increasingly unlikely that an episode ever makes it past the corridor. So, even though exploration is a necessity for innovation, it can also sabotage a learning process, well beyond being an additional work item.

In reinforcement learning, some problems are in even more need of proper exploration than others. This is specifically the case if the reward function of the problem is shaped in at least one of two problematic ways: *flat* or *deceptive*.

Take, for example, a simple gridworld, sketched in Figure 2.3. An agent starts at a central point, indicated with a circle, and is able to move to an adjacent square. There are two terminal states, one in the lower left with a small reward marked with a small X and one in the upper right with a high reward marked with a large X. Clearly, this problem requires some exploration, taking different paths in the gridworld to find the two possible terminal states to the game and ensure that there is no third solution.

The first problematic setups are *flat* (or *sparse*) rewards, meaning that the reward signal provided to the learning algorithm contains large areas that are constant. Thus, the reward function is flat, or the signal provided by the reward function is sparse. If the learning algorithm is in these areas, it is very hard for gradient-based approaches to leave these areas because the reward signal provides no information. In a maze as depicted in Figure 2.3, where the only reward is given when finding the goal, this is the case. As long as a learning algorithm has not accidentally found the goal and with it the reward signal, there is no way to learn or understand the task at hand.

Ideally, the environment is set up with some reward signals that draw the agents towards the goals. It is tempting to augment a reinforcement learning problem with secondary goals that remove the flatness from the reward and shepherd the learning process towards a desired result. While this process of *reward shaping* is a very powerful tool to improve the outcome of reinforcement learning applications, it also carries risks: It may inject the designer's presupposed notions about the problem into the solution, which goes against the aspiration

of machine learning to discover solutions independently of the ideas of the designer. But it also may evoke unintended side effects in the form of *deceptive rewards*.

Deceptive rewards are the second problematic setup, where the reward signal draws the learning algorithm to an undesired solution that is not the global solution, and that is not a proper stepping stone to reach the global solution. These are instances of proper non-convexity in the optimisation target. Such a deceptively good solution would represent a local optimum that is not the global optimum. In the maze setup, this could be represented by giving a reward signal based on the actual distance to the goal, disregarding obstacles that are on the way. Now, an agent could get stuck in a dead end that is close to the goal, but is walled off, where a global optimum would lead the agent along another path away from the pull of the local optimum that actually reaches the goal.

To differentiate the usual requirement for exploration from this additional necessity for exploration, setups with flat reward landscapes or deceptive rewards are called *hard exploration problems*.

Exploration is typically the main reason for randomness in reinforcement learning. Many learning algorithms infuse randomness into the behaviour of its agent such that it may try out different actions and encounter different states than those it has already seen. Many environments add at least some minor element of randomness in the initialisation of the environment to increase exploration. This often stabilises the results, but can also lead to problems when reporting the performance of a learning algorithm. Due to the random influences, the results of one optimisation run can differ significantly just from the random seed that initialises random number generation.

2.4 The State of Reinforcement Learning

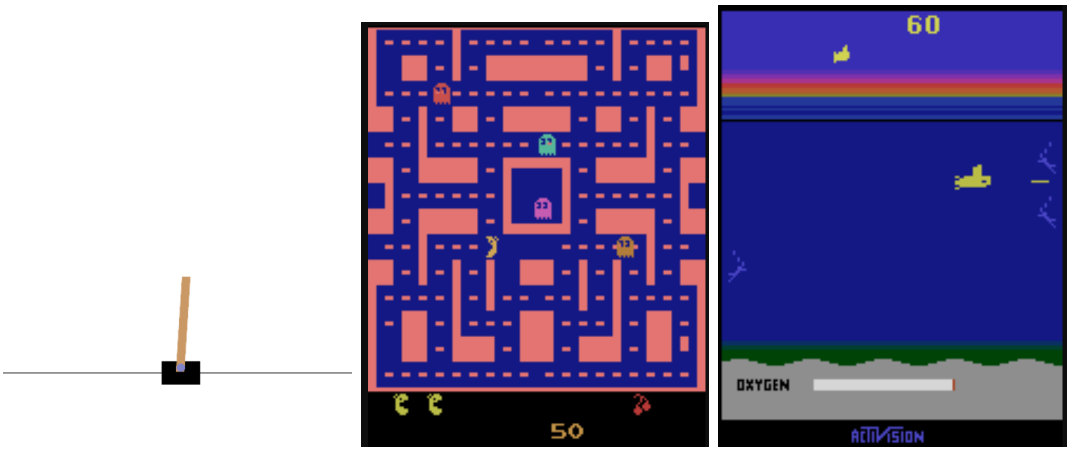
With the recent success stories around deep learning that are based on the effectiveness of large and difficult-to-understand neural networks, theoretical results garner less attention than impressive practical applications. Algorithmic improvements are engineered, and stabilising techniques do not necessarily need a theoretical backing to be incorporated into working algorithms as long as the learning is successful.

Progress in reinforcement learning is mostly measured in new and exciting benchmarks and algorithms that find solutions to beat these benchmarks. This leads to very practical results that can inspire even the general public in recent breakthroughs in playing games such as chess [SHS+17], StarCraft [VBC+19], or Dota 2 [BBC+19]. At the same time, the missing theoretical foundation can lead to results that overstate the impact of the learning approach, particularly because sometimes there exist no baseline algorithms that tackle the same problem set.

The fluidity of benchmarks can also lead to another confusion: In a reverse application of the “no free lunch” idea, there may always be a niche problem with a specific configuration, where a new suggested heuristic outperforms other benchmark algorithms. This way, unsound approaches may proliferate because there is no definitive test to find their merit.

2.4.1 Baseline Problems

Progress in reinforcement learning is often measured not in theoretical error bounds but in performance on benchmark problems. One reason for this paradigm is apparent in the introduction of the *Trust Region Reinforcement Learning* algorithm *TRPO* [SLA+15]. While the authors give an algorithm that guarantees improvement of the agent’s performance in



(a) CartPole **(b)** Atari: Ms Pacman **(c)** Atari: Seaquest

Figure 2.4: Depictions of the CartPole environment and two Atari environments.

every update step, they immediately state that the required step size is too small for practical application. The problems targeted by reinforcement learning are too large, complex, or chaotic, and the available computing time and experience in the environment are too small. Algorithms are stated using heuristics that exploit educated guesses with proofs of convergence coming later, as with the *Proximal Policy Optimization* algorithm *PPO* [SWD+17] stated in 2017, with a proof of convergence only years later [HGA+21]. Ultimately, convexity cannot be assumed anyway for most practical applications, thus reliable convergence to an optimum may not even be the property that is most sought after in reinforcement learning algorithms.

The vacuum created by the departure from mathematical proofs of convergence or stability is filled with baseline problems that represent challenges that have been successfully addressed with reinforcement learning methods in the past or that the community aspires to solve. This makes invention in reinforcement learning a leaderboard-style competition, where improvement on the benchmark problem is the ultimate goal in introducing a novel idea. While this approach has obvious weaknesses, such as the danger of fitting algorithms to the benchmark, it has been employed to great success in modern machine learning, as with the *Modified National Institute of Standards and Technology database* (MNIST) [LBB+98] and the *ImageNet project* [DDS+09] dataset.

In a stark contrast to these classification datasets, the benchmark problems in reinforcement learning are not fixed. There are classes of problems that reached some popularity: Classic control problems that predate the current deep learning boom, robotic-inspired tasks that are implemented in physics simulations, and game emulators.

The inverted pendulum is a classic control problem [BSA83] that is implemented in the *CartPole* environment (see Figure 2.4(a)). It describes a cart that moves along a frictionless track. A pole is mounted on it with a hinge. The goal is to balance the pole as long as possible without dropping it below a certain angle and without moving the cart too far from its initial position. The state space is 4-dimensional, describing the position of the cart, the angle of the pole, and the derivatives of both quantities. The action space is discrete, to either move to the left or to the right. The reward is defined only through survival. Every

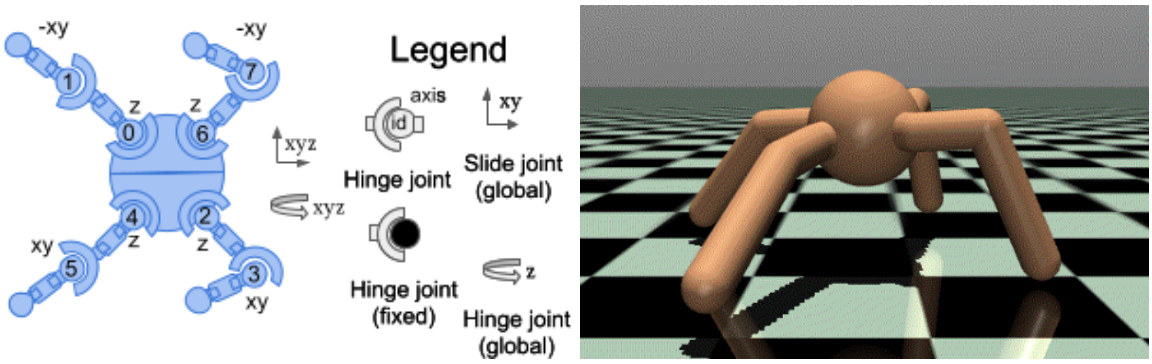


Figure 2.5: A schematic depiction of the Ant environment from the Gymnasium documentation [TKT+24] and a screenshot of a depiction of its implementation.

time step that does not violate the constraints, one point is rewarded. A time horizon of 500 time steps is set.

The quadruped walker [SML+16] implemented as the *Ant* environment is a typical representative of robotic tasks for reinforcement learning, see Figure 2.5. It is implemented in a physics simulation environment. A spherical main body is connected to four identical legs. The body is fully rigid. It is moved by actuating the joints of the legs, either the connection of the leg to the body or the middle part of the legs, the "ant's knees". A main part of the reward function is given by movement towards a certain direction, but other elements may be included as well. The actual details of the problem depend on the concrete implementation, see Subsection 2.4.3.

The *Arcade Learning Environment* features a large number of emulated games from the Atari 2600. It is an interesting use case because these games can have extremely diverse requirements for the player. Some games only require fast and correct reactions, something that is somewhat easy for a machine learner; others require planning or exploration. Due to its shared platform in the Atari 2600, the state and action spaces are the same across all the games. The state space is either defined as the RGB or the greyscale values of the screen that would be shown to a human player or as the RAM state of the gaming console. The action space consists of the 18 possible buttons or joystick interactions that the Atari offers, and one dummy action for doing nothing. The scoring system of the individual games directly defines the reward function.

New problems aim to shape the future direction of reinforcement learning research. In recent years, attention has been paid to applying reinforcement learning to different setups of the card game Hanabi [BFC+20], which features a cooperative game style and incomplete information. This represents a desire to develop cooperative agents that work well in concert, and is also mirrored in the interest in Dota 2 [BBC+19], a game played in two teams of five.

2.4.2 Baseline Random Solvers

Big AI companies are invested in reinforcement learning: The reference libraries are often implemented by departments of companies such as OpenAI [BCP+16], Meta [CB16], and Google [FFR+21]. While this increases interest and activity, it also may lead to overstatement of achievement or a focus on flashy results over solid research. Impressive results, like super-human performance in Atari games or training of robotic controllers can at least to some

extend be replicated with simple, non-deep, learning strategies as with Augmented Random Search (ARS) [MGR18] or Greedy Over Random Policy (GORP) [LRD23], which calls into question if these benchmark problems are really as complicated as they seem.

ARS defines a policy as an affine linear function that maps states to actions. It starts with randomly generated parameters $\theta \in \mathbb{R}^n$ that are iteratively updated with estimated gradient steps based on previous experiences. To get the next iteration θ' of parameters, first $2M$ new policies are created as follows. For $i = 1, \dots, M$ an n-dimensional real-valued vector $\nu_i \in \mathbb{R}^n$ is created by sampling sampling random noise $\nu_{i,j} \sim \mathcal{N}(0, \sigma)$ for some *exploration noise* parameter $\sigma \in \mathbb{R}$. Two new policies for each such vector are created, parameterised as $\pi_i^{\pm} = \theta \pm \nu_i$. Every pair of perturbations $\pi_i^{\pm}$ is then ranked by the score of the better-performing policy out of the two. The top $b < M$ perturbations according to this ranking are selected. They contribute to the next iteration like an estimated gradient of the observed accumulated discounted reward:

$$\theta' = \theta + \frac{\alpha}{b\sigma_R} \sum_{i=1}^{b} \left[\eta(\pi_i^+) - \eta(\pi_i^-)\right] v_k, \tag{2.33}$$

where α is a learning rate and σ_R the standard deviation of the $2b$ observed accumulated discounted rewards $\eta(\pi_i^{\pm})$ of the selected policies according to the ranking. Finally, the mean and standard deviation of observed states are continuously calculated, and the states are normalised accordingly before the linear policy is applied.

While this setup is clearly more intricate than a fully random search, it is still very simple, and it typically finds very good linear policies for reinforcement learning problems. Its simplicity makes it a great benchmarking algorithm that can call into question both the relevance of an algorithm and of a given problem.

The GORP algorithm showcases and exploits a surprising property of a class of reinforcement learning problems: Actions that have the highest value with respect to a random policy π^{rand} often also have the highest value with respect to the optimal policy π^*. The authors even argue that many deep reinforcement learning algorithms fail for problems that do not satisfy this property.

GORP constructs a non-stationary policy starting with a random policy in the first timestep. At the i-th timestep, given a state s_i, it repeatedly takes some steps in the environment starting with that state. It estimates a non-stationary Q-function Q_i based on the observed cumulative reward, and the policy is set to pick a maximising action regarding this action-value function. By construction, GORP is limited to discrete action spaces and deterministic Markov decision processes that can be reset to any state. However, the Atari suite fits these seemingly very limiting prerequisites, and GORP achieves strong results in several games.

Baseline solvers like these offer a rare chance in reinforcement learning to ground results achieved with new algorithms or in new environments against a baseline that is not as invested in the specifics of reinforcement learning. They can give a less biased insight into how hard new problems actually are and how complicated it is to learn a solution for them.

2.4.3 One Ant Is Not like the Other

Benchmark problems are implemented in several software packages and may differ significantly. To illustrate the confusion over benchmark reinforcement learning problems, take the Ant

environment (see Subsection 2.4.1 and Figure 2.5): It is implemented in the libraries *Gymnasium* [TKT+24], *PyBullet* [CB16], and *Brax* [FFR+21]. All implementations are based on the same reference design [SML+16], but still they differ, be it due to different physics simulations, or due to design choices, such as the dimensionality of the state space, which in Gymnasium is either 27 or 107 depending on configuration, while in PyBullet it is 30.

In Brax and Gymnasium, the newest version has 8 continuous action dimensions which actuate one of the two joints on its four legs. The state dimensions are the height of the torso and a quaternion rotation of the torso, derivatives of the full position of the torso (3 dimensions) and its orientation (3 dimensions), the angles at which the joints stand and their derivatives (8 + 8 dimensions), resulting in 27 dimensions. In Gymnasium, additional $13 \cdot 6$ observation dimensions can be added by including forces that act on the 13 body parts of the robot.

The implementation in Gymnasium has versions from v0 until v5, utilising different physics engines in the backend. This constant change of benchmark problems makes actual leaderboards difficult to maintain. Brax even ships four different backends for the physics simulation at this time. *Positional* uses position-based dynamics, a physics simulation based on the position and dynamics of particles and constraints due to collisions. *Generalized* and *mjx* are more complex multiphysics simulations, which means the simulations are based on interactions of multiple physical phenomena. They are both described as similar but are implemented independently. *Spring* is a cheaper approach that simulates the rigid body parts of robots as springs. With Brax also come reference implementations of some influential algorithms, such as ARS (*Augmented Random Search* [MGR18], Subsection 2.4.2), PPO (*Proximal Policy Optimization* [SWD+17], Subsection 3.2.2), and SAC (*Soft Actor-Critic* [HZA+18], Subsection 3.1.5).

The reward function of the Ant environment is defined as a weighted sum of four components: a flat, positive reward for being "healthy", that is if the torso of the ant stays at least at a certain height, a reward for moving towards the target direction, a negative reward for actuation which represents energy cost, and finally a negative reward for high forces acting on the robot to deter behaviour that may be dangerous to an actual robot because the forces acting on the joints may be too strong. To measure the success of a learning algorithm in the environment, the policy's performance, that is, the accumulated reward over an episode with a fixed time horizon $\sum_{i=0}^{t_{\text{term}}} r_i$, is empirically measured by rolling out an episode with that policy.

Figure 2.6 illustrates the difference a change in physics backends can cause: It shows the learning progression of three different algorithms in the Brax Ant environment with the four different physics backends. The graphs have been created by averaging over three training runs with different seeds for random number generation during training for each environment and backend. The physics simulation is theoretically deterministic. Only imprecision in computation, for example, when using GPUs, may introduce some randomness. But, the initial position of the simulated robot is randomised over a distribution of slightly different configurations of standing still. Also, all three algorithms have aspects of randomisation to encourage exploration. That makes the learning process stochastic, which can be controlled to some extent by reporting average results.

Each of the three algorithms works iteratively by varying one policy. The graphs show the performance of this iteratively changing policy on the y-axis plotted against the number of

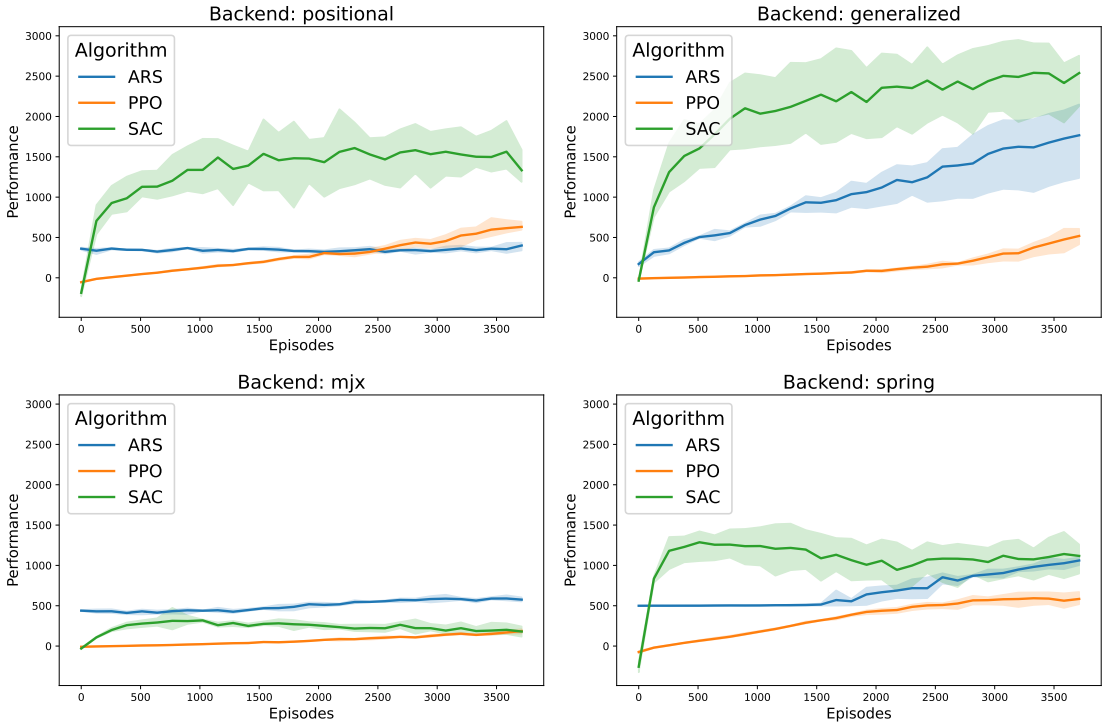


Figure 2.6: Comparison of the performance of three different reinforcement learning algorithms (ARS, PPO, SAC) in the same Ant environment on the four different backends provided by Brax. Performance is defined as the accumulated reward over one finite-time episode. The lines show the mean value when repeating the same learning process with three random seeds. The coloured area around shows the 95% confidence interval, see Appendix A.3. Even if it is four times the same problem and the same learning algorithms, the performance curves look very different, which implies that the actual problem statement changes significantly with the physics backend.

episodes played to learn from. This presentation favours sample-efficient algorithms. Graphs that stay almost completely constant, such as ARS in the *positional* backend, indicate that no meaningful learning happens. A rising curve that becomes constant, such as SAC in the generalised environment, indicates learning that has finished. A curve that keeps rising indicates that the learning process has been terminated before it has reached a locally optimal behaviour.

Both the best result achieved and the relative comparison of the three algorithms change fundamentally with the physics backend. Each of the graphs in Figure 2.6 tells a different story. If just one of these graphs is used to compare the three different algorithms, the conclusions would be different for all four. Especially notable is that the supposedly similar backends *generalized* and *mjx* induce such different results. A bachelor's thesis [Hel21] also analyses this effect over several implementations of the same problem definition.

This alone may not be critical, as researchers comparing different algorithms use the same version of a benchmark problem when preparing a publication, as in this work, too. But, in reinforcement learning, hyperparameters and even the choice of random seed matter significantly in the outcome of the learning process [HIB+18]. So even though there are

several benchmark implementations of problems and algorithms, reinforcement learning is in the replication crisis.

2.5 Reinforcement Learning in a Nutshell

To summarise, reinforcement learning is a machine learning paradigm concerned with optimal control in systems that evolve in discrete time steps, where optimality is defined by the accumulation of a reward signal. The mathematical framework in which the problem is stated is that of a Markov decision process. This mathematical framework guarantees that the reinforcement learning problem, that is searching for an optimal policy, is well posed, and optimisation algorithms to find an optimal policy can be derived from it.

An important implicit assumption in reinforcement learning is that the environment is accessible but not transparent and that learning comes through interaction with it. This makes reinforcement learning very flexible, but also difficult. It requires learning from experience and exploration. This leads to the exploration-exploitation dilemma, which is still an open problem.

The assumption that a reinforcement learning algorithm should deal with environments that are accessible but not transparent is also connected to the practice of measuring success based on observable performance in complicated problems with little regard to theoretical bounds and guarantees. This is both a strength of the field, as it leads to impressive results that capture public attention, but also a weakness, as it reduces trust and complicates further developments.

This chapter has only explored the critic approach to reinforcement learning, the idea of searching for an optimal value function, and from there deriving a policy. The introduction of actor methods and more state-of-the-art methods is shifted to the next chapter, which is concerned with behaviour in reinforcement learning, in an attempt to show the intricate connection between these very fundamental methods in reinforcement learning and the behavioural characterisation of reinforcement learning policies.

CHAPTER 3

Behaviour in Reinforcement Learning

With the basic ideas of reinforcement learning defined, the way is clear to examine behavioural differences in the theory and practice of reinforcement learning. But what is behaviour in reinforcement learning policies? In short, behaviour signifies what the policy *does* and excludes the mechanisms that arrive at the decisions which guide its actions.

Borrowed from the language of evolutionary algorithms, an algorithmic function can be characterised by its *genotype* or its *phenotype*. The genotype is a shorthand for the architecture of the function and its parameterisation. The phenotype describes the actual mathematical function, the relation between the domain and codomain that it establishes. Both terms are not strictly delimited, but rather refer to the abstract concept of looking either at the inner or outer workings of an entity. The phenotype is a result of the genotype. Inevitably, there is a strong and potentially complex connection between them. Given a fixed architecture of the parameterised function, the genotype can also be synonymous with the parameters.

How the genotype manifests in the phenotype as such is a rich subject of study that also occupies geneticists in its biological counterpart [RSP+22]. The relationship can be chaotic: small changes in the genotype can lead to significant and qualitative changes in the phenotype. The relationship is not reversible: different genotypes can lead to the same phenotype. There may exist *pleiotropy*, the phenomenon that a change at one point of data in the genome may change multiple points of data in the phenotype. This dichotomy may also manifest under different names. In deep learning, neural networks are analysed based on representational similarity (their genotype) and functional similarity (their phenotype). In the context of reinforcement learning, a phenotype is how an agent interacts with its environment. In other words, it is the agent's behaviour.

The focus on behaviour bypasses the problems of genotype-phenotype mapping. It properly enables the comparison of functions with different architectures. It also introduces eventual randomness and opaqueness of the environment into the task of comparing policies. And while the abstract delineation of behaviour seems clear, there are competing ideas about how it can be pinned down in reinforcement learning applications.

The theory and practice of reinforcement learning rely on behaviour in several ways to define problems, prove statements, and derive algorithms. The precise description and comparison of behaviour is not usually the main focus of these discourses, but rather a means to an end. This chapter highlights the occurrence of the behavioural viewpoint on policies in the literature and contextualises its motivation and utility. There are two important aspects to that: The idea to represent a policy through its behaviour, so the *behavioural characterisation* of the policy, and, secondly, how this behavioural characterisation is used to define how different two policies are, *behavioural distance*.

The behavioural characterisation of policies is the foundation for *policy gradient* algorithms. Policy gradient algorithms are actor approaches to reinforcement learning. They derive a gradient of some energy or loss function directly on the parameterisation of the policy to improve that policy through gradient descent. Section 3.1 details what policy gradient algorithms are, and how and why the policy gradient approach relies on the behavioural characterisation of policies. This can be seen in two equivalent categories of behavioural characterisation, the episodic view which interprets a policy as a random element in the space of episodes $\mathcal{T}$ or in a time-independent interpretation as a random element in the state-action space $S \times A$.

Section 3.2 explores applications in reinforcement learning that have a practical use for a distance measure based on behaviour: In imitation learning, a new policy is developed to be close in behaviour to an expert policy. Trust regions stabilise the learning process, while constraints are a specific modification of the Markov decision process framework, and both concepts use behavioural distance to not change a policy too drastically during one step of iteration during optimisation. Finally, concepts such as novelty and diversity connected to the application of evolutionary algorithms to reinforcement learning problems are also based on a behavioural characterisation and behavioural distances between policies. They traditionally work with a very different understanding of behavioural characterisation, a gap bridged in Chapter 6.

The summary of the chapter in Section 3.3 concludes that the behavioural characterisation of policies is a core idea in reinforcement learning, because the behavioural characterisation makes the policy a tangible object that can be understood and manipulated.

3.1 Behavioural Characterisation

A very basic behavioural measure of policies is implicitly used in almost all reinforcement learning experiments. To determine whether a learning algorithm has converged, the cumulative reward is observed over iterations in the optimisation as in Figure 2.6. If it stagnates, this may indicate that the algorithm has converged. This provides an additional, practical termination criterion alongside the usual convergence of the loss function, which captures the numerical change in the optimisation target. Observing the cumulative reward collected during an episode under a policy is an example of a behavioural characterisation of that policy. That is, understanding the policy based on its phenotype, its observed behaviour, instead of its genotype, the weights of the network, where change could be measured by observing a loss function. Performance is a very reductive interpretation of the full behaviour displayed. Different strategies may lead to the same score after all.

For a general approach that captures a policy by its behaviour, it is necessary to derive a more structured definition of observable behaviour and analyse its relationship to the policy. Indeed, when examining reinforcement learning theory more closely, it becomes clear that observable behaviour is not just a product of a policy. For policy gradient algorithms, a behavioural characterisation of policies is actually the basis for deriving theoretical guarantees and practical algorithms.

Understanding a policy as a function $\pi : S \to A$ is a complete behavioural characterisation, as it describes the behaviour of the policy without any ambiguity. For the practical application of distinguishing policies, this approach has one major flaw: It does not factor in how likely a policy is to encounter a certain state. This is a particularly important aspect as the behaviour of the policy influences which states are reached. Policies may even be different

in unreachable states only. For example, in chess, it is not trivial to evaluate if a certain board state is reachable at all in the game. The definition of a policy for simplicity often includes every possible game state, even if it cannot be reached by legal moves. Also, some states can only be reached after certain previous decisions. If an agent always chooses one particular opening in a game of chess, it is not really relevant how it would behave in a different opening that it never chooses.

To resolve this issue, a policy can be described by its observable behaviour. This seems to clash with Bellman's principle of optimality, which states that an optimal policy chooses an optimal action at any state. However, it becomes fundamental when the Markov decision process is defined to include a distribution of starting states. The optimisation problem changes to searching for a locally optimal policy, which is a policy optimal under these starting states [SMS+99]. Of course, starting states can be picked as a uniform distribution over all states to enforce global optimality, see also Subsection 2.3.4 on exploring starts. Actually, the two behavioural characterisations introduced in this section will be shown to be equivalent to the behavioural characterisation as a function $\pi : S \to A$ when disregarding unreachable states or constructing the Markov decision process in a way that there are no unreachable states under π.

3.1.1 Episodic Characterisation of Policies

An immediate approach to gather observable behaviour of a policy is collecting episodes of it in the environment. Given a Markov decision process and a policy for it, this policy can be empirically described as a random element where the sample space is the space of episodes in the Markov decision process. Rolling out the policy means sampling the random element. This is formalised in Theorem 1.

That means a Markov decision process with a distribution of starting states and a policy together form a random element that samples from the space of episodes. Since a policy is only defined in conjunction with a Markov decision process, if the definition of initial states is added to the idea of a Markov decision process, it justifies the use of the shorthand of π as a random element:

Definition 14 (Sampling an episode)**.** *Given a Markov decision process (S, A, p, γ, R, D) with finite event horizon and a policy π,* sampling *an episode $\tau = (s_i, a_i)_{i=1,\dots,N}$ (or* rolling out *the policy), describes the process of sampling a starting state $s_0 \sim D$, and recursively sampling $a_i \sim \pi(s_i)$ and $s_{i+1} \sim p(s_i, a_i)$ until a terminal state is reached. For consistency, also sample an action for the terminal state.*

A short-hand notation is $(s_i, a_i)_{i=1,\dots,N} = \tau \sim \pi$. For a sampled episode, the associated reward is always defined as $r_t = R(s_i, a_i, s_{t+1})$.

This characterisation of a policy as a random element on the space of episodes can justifiably be called the *episodic characterisation* of a policy. This characterisation is the basis for defining the performance of an agent as the expected cumulative discounted reward when starting an episode, or the value function sampled from a starting state, as in Equation (2.6):

Definition 15 (Performance under Starting States)**.** *Given a Markov decision process*

(S, A, p, γ, R, D) and a policy π, define the policy's performance $\eta_D(\pi)$ *as the expected value when sampling in a starting state $s \sim D$:*

$$\eta_D(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right] \tag{3.1}$$

$$= \mathbb{E}_{s_0\sim D, a_t\sim\pi(s_t), s_{i+t}\sim p(s_t,a_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t, s_{t+1})\right] \tag{3.2}$$

$$= \mathbb{E}_{s\sim D}[V^\pi(s)] \tag{3.3}$$

If the context is clear, drop the D as $\eta = \eta_D$.

An *optimally performing policy*, that is, a π^* maximising η_D, is actually only *locally optimal*, a weaker condition than an optimal policy in the original sense. Theorem 3 requires optimality in all states, not just when starting from initial states. From Bellman's principle of optimality follows that for an optimally performing policy, any state reachable by that policy will select optimal actions. However, there are no conditions on the policy's behaviour in unreachable states, regardless of whether these states are unreachable because the policy avoids them or because the setup of the problem prohibits reaching them. Tacitly, reinforcement learning algorithms that learn from experience search for an optimally performing policy, not for an optimal policy.

3.1.2 Policy Gradient on Episodes: REINFORCE

This view on behavioural characterisation is useful to derive algorithms that search for optimal policies. It is directly employed by *policy gradient* methods that work by iteratively improving the performance of a parameterised policy π^θ with a fixed architecture using gradient steps.

Policy gradient methods can be characterised as actor-only methods or be part of an actor-critic approach. The weights θ of a policy function π^θ are varied with each iteration to improve the performance. To find a gradient that guides θ towards an improvement of the policy, the performance function of the policy is maximised.

To maximise the performance of π^θ in a Markov decision process $\mathcal{M} = (S, A, p, \gamma, R, D)$, consider the performance of a policy from an episodic viewpoint as in Equation (3.1) like $\eta(\pi^\theta) = \mathbb{E}_{\tau\sim\pi^\theta}[\sum_{t=0}^{\infty}\gamma^t r_t]$. This definition of performance is based on an expected value over the random variable that samples episodes. It is based on the observable behaviour of the policy.

For simplicity, assume that both the state space and the action space are finite and that $\mathcal{M}$ has a time horizon of t_{term}. Then the space of episodes $\mathcal{T}$ is finite. Write for short $\mathcal{R}(\tau) = \sum_{t=1}^{t_{\text{term}}}\gamma^t r_t$. Let $\mathbb{P}$ be the probability measure for episodes defined in Theorem

Algorithm 1 REINFORCE algorithm

Initialise θ randomly.
for each episode **do**
  Sample episodes $\tau_i \sim \pi^\theta$ for $i = 1, \ldots, M$.
  Estimate the gradient $\nabla_\theta\eta(\pi^\theta)$ with $\{\tau_i\}_{i=1,\ldots,M}$ based on Equation (3.16).
  Update $\theta \leftarrow \theta + \alpha\nabla_\theta\eta(\pi^\theta)$.

1. Then the performance can be rewritten as $\eta(\pi^\theta) = \sum_{\tau \in \mathcal{T}} \mathbb{P}(\tau|\theta)\mathcal{R}(\tau)$, the expected discounted return when sampling an episode τ in the given environment with the policy parameterised with θ. To use that in a gradient descent algorithm, a formulation of this expression's gradient is necessary. It can be derived with some simple reformulations and the *log derivative trick*: Since $\nabla \log f = \frac{\nabla f}{f}$, it follows that $\nabla_\theta \mathbb{P}(\tau|\theta)$ can be expressed as $\mathbb{P}(\tau|\theta)\nabla_\theta \log \mathbb{P}(\tau|\theta)$ to get $\mathbb{P}(\tau|\theta)$ back and find a formulation that is encapsulated by the expected value over sampling an episode.

$$
\begin{aligned}
\nabla_\theta \eta(\pi^\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi^\theta}\left[\mathcal{R}(\tau)\right] && \text{by definition of } \eta, && (3.4)\\
&= \nabla_\theta \sum_\tau \mathbb{P}(\tau|\pi^\theta)\mathcal{R}(\tau) && \text{by definition of } \mathbb{E}_{\tau\sim\pi^\theta}, && (3.5)\\
&= \sum_\tau \nabla_\theta \mathbb{P}(\tau|\pi^\theta)\mathcal{R}(\tau) && \text{moving } \nabla_\theta, && (3.6)\\
&= \sum_\tau \mathbb{P}(\tau|\pi^\theta)\nabla_\theta \log \mathbb{P}(\tau|\pi^\theta)\mathcal{R}(\tau) && \text{with the log derivative trick,} && (3.7)\\
&= \mathbb{E}_{\tau\sim\pi^\theta}\left[\nabla_\theta \log \mathbb{P}(\tau|\pi^\theta)\mathcal{R}(\tau)\right] && \text{by definition of } \mathbb{E}_{\tau\sim\pi^\theta}. && (3.8)
\end{aligned}
$$

From this classic formulation, the expected value needs to be resolved for a practical algorithm. The most straight forward approach, to sample episodes and average over them, leads to the *REINFORCE* algorithm (Algorithm 1) [Wil92]: According to Theorem 3, the probability $\mathbb{P}(\tau|\pi^\theta)$ of encountering a specific episode τ with the policy π^θ is

$$
\begin{aligned}
\mathbb{P}(\tau|\pi^\theta) &= \prod_{t=0}^{t_{\text{term}}} \mathbb{P}(S_0 = s_0)\mathbb{P}(A_t = a_t|s_t)\mathbb{P}\left(S_{t+1} = s_{t+1}|(s_t, a_t)\right) && (3.9)\\
&= D(s_0) \prod_{t=0}^{t_{\text{term}}} \pi^\theta(a_t|s_t)p(s_{t+1}|s_t, a_t). && (3.10)
\end{aligned}
$$

The contributions of D and p are assumed to be unknown in the typical reinforcement learning setup. Consequently, their contribution is estimated from a sampled episode τ:

$$
\mathbb{P}(\tau|\pi^\theta) \approx \left[\prod_{t=0}^{t_{\text{term}}} \pi^\theta(a_t|s_t)\right]_{\tau\sim\pi^\theta} \tag{3.11}
$$

This is a sound approach, as they are both independent of θ. Therefore, sampling episodes constitutes an unbiased estimator. Then, putting this all together leads to:

$$\nabla_\theta \eta(\pi^\theta) = \mathbb{E}_{\tau\sim\pi^\theta}\left[\nabla_\theta \log \mathbb{P}(\tau|\pi^\theta)\mathcal{R}(\tau)\right] \qquad \text{by Equation (3.8),} \tag{3.12}$$

$$\approx \left[\nabla_\theta \log\left(\prod_{t=0}^{t_{\text{term}}} \pi^\theta(a_t|s_t)\right)\mathcal{R}(\tau)\right]_{\tau\sim\pi^\theta} \qquad \text{by Equation (3.11),} \tag{3.13}$$

$$= \left[\sum_{t=0}^{t_{\text{term}}} \nabla_\theta \log \pi^\theta(a_t|s_t)\mathcal{R}(\tau)\right]_{\tau\sim\pi^\theta} \qquad \text{by the logarithm product rule,} \tag{3.14}$$

$$= \left[\sum_{t=0}^{t_{\text{term}}} \nabla_\theta \log \pi^\theta(a_t|s_t) \sum_{\bar{t}=0}^{t_{\text{term}}} \gamma^t r_t\right]_{\tau\sim\pi^\theta} \qquad \text{by definition of } \mathcal{R}. \tag{3.15}$$

Now, estimating the behaviour of the environment not through one, but instead $M \in \mathbb{N}$ episodes $\tau_i = (s_{i,t}, a_{i,t})$ for $i = 1, \dots, M$, leads to the following estimation:

$$\nabla_\theta \eta(\pi^\theta) \approx \frac{1}{M}\sum_{i=1}^{M}\sum_{t=0}^{t_{\text{term}}} \nabla_\theta \log \pi^\theta(a_{i,t}|s_{i,t}) \sum_{\bar{t}=0}^{t_{\text{term}}} \gamma^{\bar{t}} r_{i,\bar{t}} \tag{3.16}$$

With this gradient estimate, an iterative algorithm using stochastic gradient descent can be formulated in Algorithm 1.

The formulation through a stochastic gradient descent implies that with a sufficiently small learning rate, this algorithm should at least converge to a local optimum. The precise properties of even this simple algorithm are still under investigation. A recent result finds that for discrete and fully reachable state and action spaces with a time horizon and a specific policy architecture, REINFORCE converges for any learning rate [MCD+25].

3.1.3 Occupancy Measure as Characterisation

Keeping in mind the Markov property, both of the Markov decision process and of any optimal policy, the episodic characterisation of policies seems out of place. After all, the evolution over time captured in an episode is not really necessary when defining the optimal policy theoretically or when searching for an optimal policy in practice. In fact, it is possible to derive from the episodic characterisation an even more concise characterisation where the temporal component can be dropped with the *occupancy measure* (also sometimes *occupation measure*):

Definition 16 (Discounted occupancy measure)**.** *Let π be a policy for a Markov decision process (S, A, p, γ, R, D), $\mathbb{P}$ be the derived probability measure as in Theorem 1 with the random elements $(S_i, A_i)_{i\in\mathbb{N}_0}$.*

$$d_\pi(s, a) := (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \mathbb{P}(S_t = s, A_t = a) \tag{3.17}$$

is the discounted occupancy measure *of* π. *Given a time horizon* t_{term}, *it is also possible to define a non-discounted occupancy measure* ($\gamma = 1$) *as*

$$d_\pi(s,a) := \frac{1}{t_{term}} \sum_{t=0}^{t_{term}-1} \mathbb{P}(S_t = s, A_t = a). \tag{3.18}$$

Theorem 4. *The discounted occupancy measure* d_π *is a probability measure.*

Proof. Non-negativity d_π is assured, as each summand and factor is non-negative. Countable additivity follows from the countable additivity of each summand. Normalisation is left to show. This follows from $\int_{S\times A} \mathbb{P}(S_t = s, A_t = a)\, ds\, da = 1$ with

$$\int_{S\times A} d_\pi(s,a)\, ds\, da \tag{3.19}$$

$$= (1-\gamma) \sum_{t=0}^{\infty} \int_{S\times A} \gamma^t \mathbb{P}(S_t = s, A_t = a)\, ds\, da \qquad \text{by definition of } d_\pi, \tag{3.20}$$

$$= (1-\gamma) \sum_{t=0}^{\infty} \gamma^t, \qquad \text{since } s \in S \text{ and } a \in A, \tag{3.21}$$

$$= \frac{1-\gamma}{1-\gamma} = 1. \tag{3.22}$$

□

Figure 3.1 illustrates how the episodic characterisation and the occupancy characterisation present a policy by observing the policy's behaviour in a trivial Markov decision process with a trivial policy.

The occupancy measure is, in some sense, a unique behavioural representation of a policy. It clearly defines how to act in all states that are reachable by the policy and from that, functionally, the policy can be reconstructed. To show this, the *Bellman flow constraints* must first be defined.

Definition 17 (Bellman flow constraints)**.** *Let* (S, A, p, γ, R, D) *be a Markov decision process with finite state and action space. The non-negative function* $f : S \times A \to \mathbb{R}_+$ *fulfils the* Bellman flow constraints *if for each* $s \in S$ *and* $a \in A_s$, $f(s,a)$ *satisfies*

$$\sum_{a\in A_s} f(s,a) = D(s) + \gamma \sum_{\tilde{s}\in S, a\in A_s} f(\tilde{s},a) p(s|\tilde{s},a). \tag{3.23}$$

In words, $f(s,a)$ is a quantity for each $s \in S$ and each $a \in A$, larger than or equal to zero. When integrated over all actions a in the given state s, it is exactly the probability of that state to be a starting state plus the discounted probability of that state being reached from any other previous state $\tilde{s}$ performing the action a, represented by weighing $f(\tilde{s},a)$ with the probability $p(s|\tilde{s},a)$ of transitioning to the state s.

Theorem 5 (Identification of policies and occupancy measure ([SBS08]))**.** *Let* $\mathcal{M} = (S, A, \gamma, p, R, D)$ *be a Markov decision process with finite state and action space. Let* f *fulfil the Bellman flow constraints and* $\pi(a|s) = \frac{f(s,a)}{\sum_{a\in A_s} f(s,a)}$. *Then* π *is a policy and* f *is its*

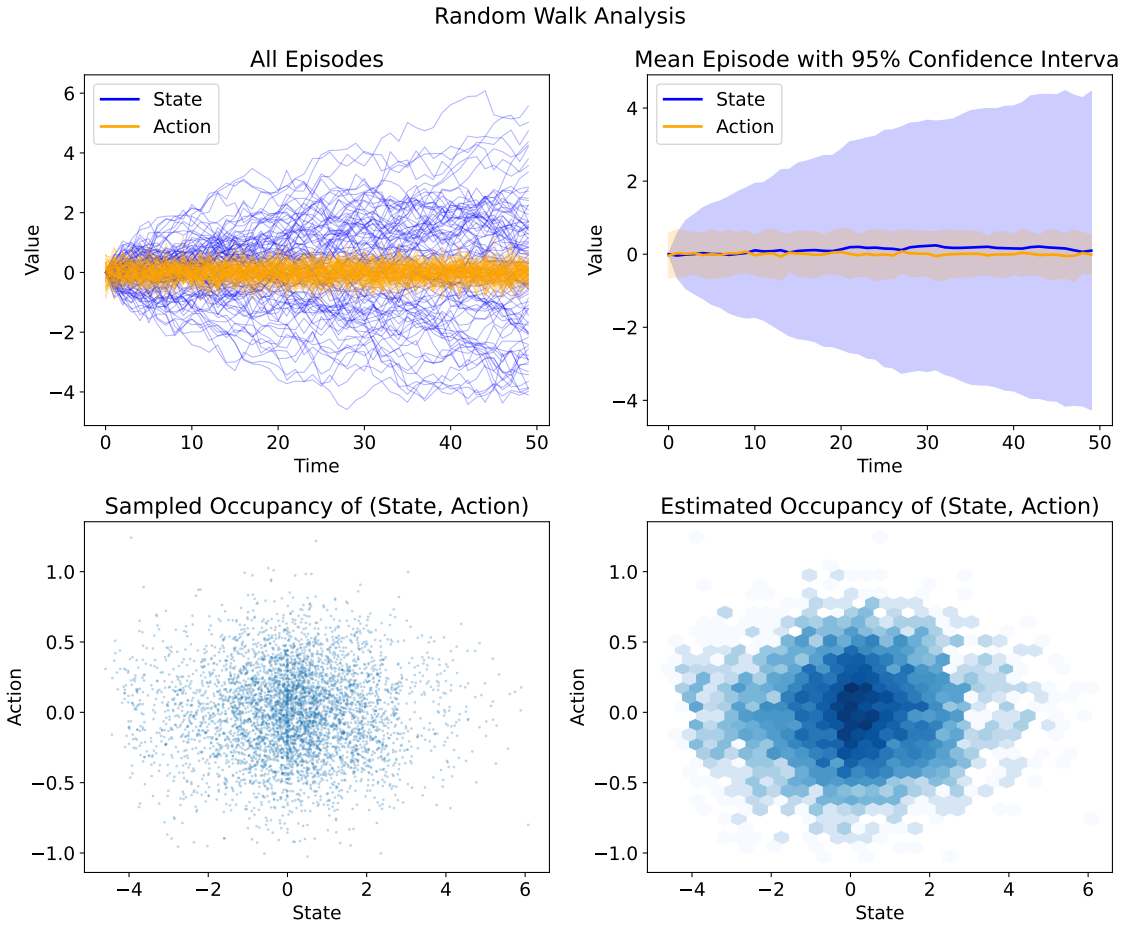


Figure 3.1: Let a Markov decision process be defined with $S, A = \mathbb{R}$, $T(s,a) = s + a$, a time horizon of 50 time steps and $s_0 = 0$ as the only starting state. Let π be a policy that picks at random from the normal distribution $\mathcal{N}(0, 0.1)$. The four pictures depict the observed episodic characterisation (top) and occupancy characterisation (bottom) through samples (left) and mean values (right).

occupancy measure.

Let π be a policy in $\mathcal{M}$ and let f be its occupancy measure. Then $\pi(a|s) = \frac{f(s,a)}{\sum_{a \in A_s} f(s,a)}$ and f satisfies the Bellman flow constraints.

Proof. See [SBS08]. □

This characterisation can be used for a similar derivation as the episodic characterisation. For example, the performance of a policy π can be represented through the occupancy measure.

Theorem 6 (Performance through occupancy)**.** *Let $\mathcal{M}$ be a Markov decision process with starting state distribution D and discount factor γ. Let π be a policy for $\mathcal{M}$ and d_π the discounted occupancy measure of π. The performance $\eta(\pi)$ of that policy can be represented*

as

$$\eta(\pi) = \frac{1}{1-\gamma}\mathbb{E}\left[R(s,a,s') \mid (s,a) \sim d_\pi, s' \sim p_{(s,a)}\right]. \tag{3.24}$$

Proof. Let R_t be the random elements that describe the reward in time step t under the policy π and the Markov decision process $\mathcal{M}$:

$$\eta(\pi) = \mathbb{E}\left[V^\pi(s_0) \mid s_0 \sim D\right] \qquad \text{by Definition 15,} \tag{3.25}$$

$$= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right] \qquad \text{by Definition 11.} \tag{3.26}$$

Since $\mathbb{E}\left[\left|\sum_{t=0}^{\infty} \gamma^t R_t\right|\right]$ converges,

$$= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}\left[R_t\right] \tag{3.27}$$

$$= \frac{1}{1-\gamma}\mathbb{E}\left[R(s,a,s') \mid (s,a) \sim d_\pi, s' \sim p(s,a)\right] \qquad \text{by Definition 16.} \tag{3.28}$$

□

A similar idea compares the difference in performance between two policies. This is particularly useful for learning algorithms because it enables an estimation of the change in performance when changing the policy without actually rolling the new policy out (see Subsection 3.2.2).

Theorem 7 (Advantage over support ([KL02]))**.** *For a Markov decision process* (S,A,p,γ,R,D) *with policies* $\pi, \tilde{\pi}$*, the difference in their performance can be represented by the advantage* A^π *(see Definition 12) of one policy over the other on just the other's occupancy measure:*

$$\eta_D(\tilde{\pi}) - \eta_D(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim d_{\tilde{\pi}}}\left[A^\pi(s,a)\right] \tag{3.29}$$

Proof. Let S_t, A_t be the random elements that for $\tilde{\pi}$ describe the probability of being in state s with action a respectively at the t-th step as described in Theorem 1 and let R_t be the random element for the reward at time step t. The proof shows the relationship between the expected rewards and the occupancy measure, which can be identified with the idea of transitions in reinforcement learning.

$$\eta_D(\tilde{\pi}) = \sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t R_t\right] \qquad \text{By definition of } \eta_D \tag{3.30}$$

$$= \sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t (R_t + V^\pi(S_t) - V^\pi(S_t))\right] \qquad \text{Adding 0} \tag{3.31}$$

Now, by manipulating $\sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t V^\pi(S_t)\right]$, pulling out the first summand

$$\sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t V^\pi(S_t)\right] = \mathbb{E}_{s_0\sim D}[V^\pi(s_0)] + \sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^{t+1} V^\pi(S_{t+1})\right] \tag{3.32}$$

and plugging that back into Equation (3.31), this resolves to

$$\eta_D(\tilde{\pi}) = \sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t (R_t + \gamma V^\pi(S_{t+1}) - V^\pi(S_t))\right] + \mathbb{E}_{s_0 \sim D}[V^\pi(s_0)]. \tag{3.33}$$

Using the definition of Q^π and η_D,

$$= \sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t \left(Q^\pi(S_t, A_t) - V^\pi(S_t)\right)\right] + \eta_D(\pi), \tag{3.34}$$

the definition of A^π,

$$= \sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t A^\pi(S_t, A_t)\right] + \eta_D(\pi), \tag{3.35}$$

and the definition of $d_{\tilde{\pi}}$

$$= \frac{1}{1-\gamma} \mathbb{E}_{(s,a)\sim d_{\tilde{\pi}}}\left[A^\pi(s,a)\right] + \eta_D(\pi), \tag{3.36}$$

which concludes the proof. □

These formulations already hint that the concept of the occupancy measure may be useful. Indeed, it emerges in many concrete reinforcement learning algorithms, mainly through some variant of the *policy gradient theorem*.

3.1.4 Policy Gradient Theorem

There is another, more practical formulation of the policy gradient method than the one that leads to the REINFORCE algorithm. It is based on shifting from an episodic understanding of the policy to an understanding based on its occupancy. Similar to Q-learning (see Subsections 2.3.2 and 2.3.3), this enables learning on transitions (s, a, r, s') instead of episodes. These formulations dominate the landscape of state-of-the-art algorithms. Their superior performance may be due to their ability to decouple and, therefore, abstract sections of an episode from the outcome of the episode as a whole. Even in a won game, there may have been bad moves, and something to learn from them.

To achieve this formulation based on the occupancy measure, the expression $\nabla_\theta \eta(\pi^\theta)$ is reformulated in similar fashion as before. This reformulation then needs more intricate calculations to arrive at a formulation that is known as the policy gradient theorem:

Theorem 8 (Policy gradient theorem ([SMS+99]))**.** *Let (S, A, p, γ, R, D) be a Markov decision process with finite state and action space, and let π^θ be a parameterised policy for it that is differentiable in $\theta \in \mathbb{R}^n$. Then*

$$\nabla_\theta \eta(\pi^\theta) = \frac{1}{1-\gamma} \mathbb{E}_{s,a\sim d_{\pi^\theta}}\left[\nabla_\theta \log \pi^\theta(a|s) Q^{\pi^\theta}(s,a)\right] \tag{3.37}$$

Proof. First note that $V^{\pi^\theta}(s) = \mathbb{E}_{a\sim\pi^\theta}\left[Q^{\pi^\theta}(s,a)\right] = \sum_{a\in A}\pi^\theta(a|s)Q^{\pi^\theta}(s,a)$ to derive:

$$\nabla_\theta \mathbb{E}_{s_0\sim D}[V^{\pi^\theta}(s_0)] = \nabla_\theta \sum_{s_0\in S} D(s_0)V^{\pi^\theta}(s_0) \tag{3.38}$$

$$= \nabla_\theta \sum_{s_0\in S} D(s_0) \sum_{a_0\in A} \pi^\theta(a_0|s_0)Q^{\pi^\theta}(s_0,a_0) \tag{3.39}$$

$$= \sum_{s_0\in S} D(s_0) \sum_{a_0\in A} \nabla_\theta \left(\pi^\theta(a_0|s_0)Q^{\pi^\theta}(s_0,a_0)\right) \tag{3.40}$$

The expression inside the inner sum is resolved with the product rule.

$$\nabla_\theta \left(\pi^\theta(a_0|s_0)Q^{\pi^\theta}(s_0,a_0)\right) \tag{3.41}$$

$$= \nabla_\theta\pi^\theta(a_0|s_0)Q^{\pi^\theta}(s_0,a_0) + \pi^\theta(a_0|s_0)\nabla_\theta Q^{\pi^\theta}(s_0,a_0) \tag{3.42}$$

To make the transition from a formulation based on starting states to the occupancy, the temporal evolution of an episode that is implicitly contained in the Q-function has to be expanded appropriately. The gradient over the Q-function can be expanded as

$$\nabla_\theta Q^{\pi^\theta}(s_0,a_0) \tag{3.43}$$

$$= \nabla_\theta \left(\mathbb{E}\left[r_0 + \gamma V^{\pi^\theta}(s_1) \mid (r_0,s_1)\sim p(s_0,a_0)\right]\right) \tag{3.44}$$

$$= \nabla_\theta \mathbb{E}\left[r_0 \mid (r_0,s_1)\sim p(s_0,a_0)\right] + \gamma\nabla_\theta \mathbb{E}\left[V^{\pi^\theta}(s_1) \mid (r_0,s_1)\sim p(s_0,a_0)\right] \tag{3.45}$$

Since it is independent of θ, the expression $\nabla_\theta \mathbb{E}\left[r_0 \mid (r_0,s_1)\sim p(s_0,a_0)\right]$ vanishes. Rewriting the second summand, exploiting the identity $V^{\pi^\theta}(s) = \sum_{a\in A} Q^{\pi^\theta}(s,a)\pi^\theta(a|s)$, gives

$$\nabla_\theta Q^{\pi^\theta}(s_0,a_0) = \gamma\nabla_\theta \mathbb{E}\left[V^{\pi^\theta}(s_1) \mid (r_0,s_1)\sim p(s_0,a_0)\right] \tag{3.46}$$

$$= \gamma\nabla_\theta \sum_{s_1\in S} V^{\pi^\theta}(s_1)p(s_1|s_0,a_0) \tag{3.47}$$

$$= \gamma\nabla_\theta \sum_{s_1\in S} p(s_1|s_0,a_0) \sum_{a_1\in A} Q^{\pi^\theta}(s_1,a_1)\pi^\theta(a_1|s_0) \tag{3.48}$$

$$= \gamma \sum_{s_1\in S} p(s_1|s_0,a_0) \sum_{a_1\in A} \nabla_\theta \left(Q^{\pi^\theta}(s_1,a_1)\pi^\theta(a_1|s_1)\right). \tag{3.49}$$

Which is the same expression as in Equation (3.40), but state and action have been progressed to the next timestep (s_1,a_1). The product $\pi^\theta(a_0|s_0)p(s_1|s_0,a_0)$ is exactly the probability of progressing to s_1 from s_0 under the Markov process that is defined by the Markov decision process under the policy π^θ. According to Theorem 1, random elements S_i encode the probability of encountering a certain state at timestep i. From this viewpoint the starting states D can be understood as S_0, and the expression $\pi^\theta(a_0|s_0)p(s_1|s_0,a_0)$ can be understood as $\mathbb{P}(S_1 = s_1|s_0)$, as the probability of that process to be in state s_1 at timestep 1, given state s_0 at the previous timestep.

Summarising, the gradient of the performance can be rewritten, first in relation to the state-

action pair of the first time step and then expanded in relation to the first two state-action pairs:

$$\nabla_\theta \mathbb{E}_{s\sim D}[V^{\pi^\theta}(s)] \tag{3.50}$$
$$= \sum_{s_0\in S} \mathbb{P}(S_0 = s_0) \sum_{a_0\in A} \nabla_\theta \left(\pi^\theta(a_0|s_0) Q^{\pi^\theta}(s_0, a_0)\right) \tag{3.51}$$
$$= \sum_{s_0\in S} \mathbb{P}(S_0 = s_0) \Bigg(\sum_{a_0\in A} \nabla_\theta \pi^\theta(a_0|s_0) Q^{\pi^\theta}(s_0, a_0) \tag{3.52}$$
$$+ \gamma \sum_{s_1\in S} \mathbb{P}(S_1 = s_1|s_0) \sum_{a_1\in A} \nabla_\theta \left(Q^{\pi^\theta}(s_1, a_1)\pi^\theta(a_1|s_1)\right) \Bigg) \tag{3.53}$$

Now, resolving the brackets, and simplifying

$$\sum_{s_0\in S} \mathbb{P}(S_0 = s_0) \sum_{s_1\in S} \mathbb{P}(S_1 = s_1|s_0) = \sum_{s_1\in S} \mathbb{P}(S_1 = s_1), \tag{3.54}$$

the contribution of the first state-action pair and of the second state-action pair are represented as separate summands:

$$\nabla_\theta \mathbb{E}_{s\sim D}[V^{\pi^\theta}(s)] \tag{3.55}$$
$$= \sum_{s_0\in S} \mathbb{P}(S_0 = s_0) \sum_{a_0\in A} \nabla_\theta \pi^\theta(a_0|s_0) Q^{\pi^\theta}(s_0, a_0) \tag{3.56}$$
$$+ \gamma \sum_{s_1\in S} \mathbb{P}(S_1 = s_1) \sum_{a_1\in A} \nabla_\theta \left(Q^{\pi^\theta}(s_1, a_1)\pi^\theta(a_1|s_1)\right) \tag{3.57}$$

The second summand resembles precisely Equation (3.40) with incremented indices and an additional factor for γ, so the same expansion can be repeated to get the infinite sum

$$\nabla_\theta \mathbb{E}_{s\sim D}[V^{\pi^\theta}(s)] \tag{3.58}$$
$$= \sum_{i\in\mathbb{N}_0} \sum_{s_i\in S} \gamma^i \mathbb{P}(S_i = s_i) \sum_{a_i\in A} \nabla_\theta \pi^\theta(a_i|s_i) Q^{\pi^\theta}(s_i, a_i) \tag{3.59}$$

Now in each summand, the log derivative trick $\nabla \log f = \frac{\nabla f}{f}$ can be applied, the sums and probabilities of state and action can be combined, $\pi^\theta(a_i|s_i)$ is replaced with $\mathbb{P}(A_i = a_i|s_i)$, and finally an expected value over the occupancy measure is formulated:

$$\nabla_\theta \mathbb{E}_{s\sim D}[V^{\pi^\theta}(s)] \tag{3.60}$$
$$= \sum_{i\in\mathbb{N}_0} \sum_{(s_i,a_i)\in S\times A} \gamma^i \mathbb{P}(S_i = s_i)\pi^\theta(a_i|s_i) \nabla_\theta \left(\log \pi^\theta(a_i|s_i)\right) Q^{\pi^\theta}(s_i, a_i) \tag{3.61}$$
$$= \sum_{(s,a)\in S\times A} \sum_{i\in\mathbb{N}_0} \gamma^i \mathbb{P}(S_i = s, A_i = a) \nabla_\theta \left(\log \pi^\theta(a|s)\right) Q^{\pi^\theta}(s, a) \tag{3.62}$$

By Definition 16, $\sum_{i\in\mathbb{N}_0}\gamma^i\mathbb{P}(S_i=s,A_i=a)=\frac{1}{1-\gamma}d_{\pi^\theta}(s,a)$. That enables the shift from an episodic to an occupancy-based viewpoint, proving the statement with

$$\nabla_\theta\mathbb{E}_{s\sim D}[V^{\pi^\theta}(s)] = \sum_{(s,a)\in S\times A}\frac{1}{1-\gamma}d_{\pi^\theta}(s,a)\nabla_\theta\left(\log\pi^\theta(a|s)\right)Q^{\pi^\theta}(s,a) \tag{3.63}$$

$$= \frac{1}{1-\gamma}\mathbb{E}_{s,a\sim d_{\pi^\theta}}\left[\nabla_\theta\log\pi^\theta(a|s)Q^{\pi^\theta}(s,a)\right]. \tag{3.64}$$

□

The policy gradient theorem is used directly or slightly adapted to derive several algorithms in reinforcement learning. By working on the occupancy of the policy, it justifies the use of replay buffers that store only transitions, instead of full episodes and then stochastic gradient descent on those transitions. The process of reformulating the expected value on an episode to a workable expected value on the occupancy measure is typically the key to other methods that follow this paradigm with changed basic assumptions, for example, in the deterministic policy gradient theorem [SLH+14].

3.1.5 Actor-Critic Algorithms

The proof of the policy gradient theorem concludes the basics of reinforcement learning. This subsection introduces three actor-critic reinforcement learning algorithms relevant to this work, either as benchmark algorithms or as building blocks of other algorithms.

An instance of a state-of-the-art algorithm that utilises a variant of the policy gradient theorem is the *Twin Delayed Deep Deterministic policy gradient* algorithm *TD3* [FHM18]. It is an actor-critic approach that approximates both an action-value function and a policy function. The action-value function is learned on transitions with the Bellman equation. The update for the actor is derived in the fashion of the policy gradient theorem, Theorem 8. When deriving the gradient, the policy is assumed to be deterministic. The same strategy of translating the episodic understanding of the policy to an understanding based on the occupancy measure then leads to the deterministic policy gradient theorem [SLH+14]

$$\nabla\eta(\pi^\theta)=\mathbb{E}_{s\sim d_{\pi^\theta}}\left[\nabla_a Q^{\pi^\theta}(s,a)_{|a=\pi^\theta(s)}\nabla_\theta\pi^\theta(s)\right]. \tag{3.65}$$

which inform the update step of the actor in TD3.

Two other practical ideas are used to improve the performance of the algorithm: Random noise provides the necessary exploration of the continuous action space, which in other strategies may be achieved with a nondeterministic policy. So instead of learning from experience that is created from actions $a=\pi^\theta(s)$, diverse experiences are created from actions $\tilde{a}=a+\varepsilon$ with $a=\pi^\theta(s)$ and $\varepsilon\sim\mathcal{N}(0,\sigma)$. That noise is clipped to be at most a certain value. TD3 also uses the idea of twin delayed learning, having two sets of parameters as weights for the action-value function from which a Bellman target is created with an estimation based on the minimal value

$$Q^{\psi_i}(s,a)-(r+\min_{i=1,2}Q^{\psi_i}(s',a')), \tag{3.66}$$

for a transition (s,a,r,s',a') with $s'\sim p_{(s,a)}$, $r=R(s,a,s')$, $a'\sim\pi(s')$, parameters ψ_i, and $i=1,2$. This counteracts the phenomenon of the *overestimation bias* where early positive

Algorithm 2 TD3: Twin delayed deep deterministic policy gradient

Hyperparamters: Discount factor γ, replay buffer size N_{buff}, critic learning rate α_{critic}, policy learning rate α_{policy}, policy noise standard deviation σ, noise clip value c, transition batch size N_{batch}, soft update parameter τ, policy update delay n_{delay}, number total iterations N_{iter}, and number of environment interactions per iteration n_{steps}.
Initialise parameters of critics ψ_1, ψ_2, actor θ, and targets $\psi_1', \psi_2', \theta'$. Initialise the replay buffer as a queue that holds N_{buff} transitions and control variable $j = 0$.
For any policy π, let $\tilde{\pi}(s) = [\pi(s) + [\varepsilon]_c]_{A_s}$, with $\varepsilon \sim \mathcal{N}(0, \sigma)$ where $[x]_c$ indicates clipping such that $x \in [-c, c]$ and $[x]_{A_s}$ clipping such that $x \in A_s$.
for $j = 1, \ldots, N_{\text{iter}}$ **do**
    Take n_{steps} steps in the environment with modified target policy $\tilde{\pi}^{\theta'}$.
    Add observed transitions to the replay buffer.
    Sample batch of N_{batch} transitions $\{(s, a, r, s')\}$ from the replay buffer.
    If $s \notin S_{term}$ let $y \leftarrow r + \gamma \min_{i=1,2} Q^{\psi_i'}(s', \tilde{\pi}^{\theta'}(s'))$, else let $y \leftarrow r$.
    Update ψ_i with loss $L(\psi_i) = (Q^{\psi_i}(s, a) - y)^2$ and learning rate α_{critic} for $i = 1,2$.
    if $j \bmod n_{\text{delay}} = 0$ **then**
        Update θ with gradient $\nabla_a Q^{\psi_1}(s, a)_{|a=\pi^\theta(s)} \nabla_\theta \pi^\theta(s)$ and learning rate α_{policy}.
        Soft update targets: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$, $\psi_i' \leftarrow \tau\psi_i + (1 - \tau)\psi_i'$ for $i = 1,2$.

experiences may derail the learning process, when the learner commits to always repeat these patterns instead of further exploration. Soft τ updates further smooth out the learning process as updates are made by setting new network weights only by blending in $\tau \in (0, 1)$ of the weights of the active learning network with $1 - \tau$ weights of the target network. For example, instead of updating $\theta' \leftarrow \theta$, a soft τ update is defined as

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'. \tag{3.67}$$

The TD3 learning algorithm iteratively improves a policy based on experience in a replay buffer, which may have been created during earlier iterations and is always created by a version of the policy with added random noise. Therefore, it is an off-policy learning algorithm, an algorithm learning from the experience that may be created with a policy different from the one being improved, see also Subsection 2.3.4.

Because TD3 is an important part of an original algorithm introduced in Chapter 6, it is fully described here in Algorithm 2.

Soft Actor-Critic (SAC) [HZA+18] is another relevant reinforcement learning strategy that also employs both of these improvements, twin delay updates and soft τ updates. Its unique feature is that it implements exploration by including the entropy $h(\pi^\theta(s))$ of the policy π^θ into the optimisation target. The entropy is defined as

$$\mathbb{E}_{a \sim \pi(s)} \left[-\log \pi(a|s) \right]. \tag{3.68}$$

The optimal policy maximises an utility function J with

$$J(\theta) = \mathbb{E}_{(s,a,r) \sim d_\pi} \left[r + \alpha h(\pi^\theta(s)) \right], \tag{3.69}$$

where the *temperature* α determines the influence of the entropy. This approach leads to the formulation of a soft value function that includes an entropy term. It is also called entropy-regularised, as this addition smooths out the value function. The iterative optimisation procedure estimates soft state-action values of a given policy. It creates the next policy through a gradient step that guides the policy towards higher soft state-action values, while still incorporating the entropy term. SAC is off-policy, because it learns from experiences in a replay buffer that a policy from earlier iterations may have created.

The *Advantage Actor-Critic (A2C)* [MBM+16] develops a gradient estimate based on the advantage function (see Definition 12) instead of a value function. This then leads to the theoretical gradient

$$\nabla\eta(\pi^\theta) = \mathbb{E}_{(s,a)\sim d_{\pi^\theta}}\left[A(s,a)\log\pi^\theta(a|s)\right]. \tag{3.70}$$

Given an observed transition (s, a, r, s'), the formulation of the advantage function $A(s,a) = Q(s,a) - V(s)$ is simplified with the estimation $Q(s,a) \approx r + \gamma V(s')$. Another learned neural network, the critic, approximates V. A2C works with a stochastic policy to address the need for exploration. It only learns from experience collected with the current policy, so A2C is an on-policy learning algorithm.

Table 3.1 gives an overview of the differences between the three algorithms introduced here. Their introduction serves as a reference for this work, as these algorithms are used throughout for different applications. But it also shows that the policy gradient theorem is a very productive basis for developing new reinforcement learning algorithms. By starting with slightly different assumptions, such as regularising the value function with entropy, a different policy gradient update can be derived, which is the heart of the iterative optimisation procedure. It also shows that some practical ideas for improving the stability and performance of reinforcement learning algorithms are shared and carried over from one algorithm to another, as with twin delay and soft τ updates.

Table 3.1: Comparison of SAC, TD3, and A2C

Feature	**SAC**	**TD3**	**A2C**
Exploration approach	Entropy maximisation	Random noise	Stochastic policy
PG approach	Soft Q-function	Deterministic	Advantage
Twin delay updates	Yes	Yes	No
Soft τ updates	Yes	Yes	No
On/off-policy	Off-policy	Off-policy	On-policy

3.2 Applications of Behavioural Distance

Behavioural characterisation is a fundamental ingredient of reinforcement learning theory that lies at the heart of policy gradient approaches. The policy gradient is estimated based on the policy's observed behaviour, and this is justified by shifting the goal of the optimisation, not to search for an optimal policy following Definition 5 but rather for an optimally performing policy following Definition 15.

To define similarity and dissimilarity of policies, it is essential to determine the subject of the investigation. If a policy is understood as a function, it is tempting to make comparisons in some function space. If the possible behaviour across all possible states is not of interest during learning, then it should probably not be of interest when analysing policies either. The results of searching for an optimally performing policy should not be analysed based on an understanding that presupposes an equal importance for all states. If the only things that matter when training the policies are the states actually encountered and the decisions actually made, the analysis of the policies should reflect that.

That suggests working with a behavioural understanding of policies. This leads to the concept of *behavioural distance*, defining the difference of policies precisely through the states they encounter and actions they execute. From Subsections 3.1.1 and 3.1.3, two main avenues are derived. One is to treat the policy as a random process that creates time series: $\tau \sim \pi$ in an episodic understanding of behaviour. The other, through the idea of occupancy, drops the time-dependency without losing information (see Theorem 5).

In fact, concepts of behavioural distance have useful applications in reinforcement learning: when learning from demonstrations in imitation learning, to stabilise the learning process with trust regions; when avoiding undesired behaviour under constraints; and when developing a population of solutions with different behaviours looking for novelty and diversity.

3.2.1 Imitation Learning

Imitation learning [ZKK+24] comprises several algorithms that create solutions to reinforcement learning problems based on the imitation of the behaviour of an expert. This is a pillar of human learning, so approaches to automate learning from demonstration are of interest. A typical test setup can be created by training an expert policy with standard reinforcement learning. The task of imitation learning is then to create another policy by learning not from the reward feedback in the environment, but just by imitating the expert's behaviour. The success of an imitation learning approach is measured by how much of the expert's performance is recovered this way. The relevance of imitation learning lies in its ability to tackle learning tasks where no reward feedback is provided and only expert demonstrations exist, for example, training robotic controllers from human demonstrations. The field can be divided into four categories: Inverse reinforcement learning, adversarial imitation learning, imitation from observation, and behavioural cloning.

Inverse reinforcement learning tries to first create a reinforcement learning problem from the expert demonstrations and then use reinforcement learning approaches to find a solution. *Adversarial imitation learning* creates an adversarial learning problem, where a discriminator labels episodes from the learner and the expert, and the learner tries to fool the discriminator. Discriminators use, for example, the Shannon-Jensen divergence, Kullback-Leibler divergence, and Wasserstein distance. *Imitation from observation* encompasses practical approaches to engineer learning algorithms from complex observations only, such as video feeds. This requires careful combination and integration of ideas from other fields of imitation learning

and other practical solutions in machine learning, such as solutions for image recognition. *Behavioural cloning* is based on a supervised learning task, where a function that maps from states to actions is learned, based on the examples of the expert. This reflects a view of a policy as a function. These techniques face the *covariate shift problem*. Even due to slightly different behaviour, the learner may encounter situations that are substantially different from the demonstrations given by the expert for which it is not trained. The *Dataset Aggregation* or *DAgger* algorithm [RGB11] is a classic interactive imitation learning algorithm that allows querying the expert to resolve this problem. Other approaches train the learner to stay within the same states as the expert, for example, by estimating the support of the occupancy measure [WCA+19].

The DAgger algorithm is interesting here, because it addresses the issues that arise when defining the difference of policies as the difference of two functions. It navigates this challenge by weighing the functional difference based on the observed behaviour. Let π^* be an expert policy, and for any policy $\pi : S \to A$ let $L(s, \pi)$ be a loss function defined as an expected difference in behaviour between π and π^* in the state s. In an environment with a small, finite action space, this, for example, could be $\mathbb{E}\left[\delta_{(\pi(s),\pi^*(s))}\right]$, the probability that π and π^* take the same action given state s. To distinguish this difference from behaviour that in a general sense describes how a policy navigates its environment, call $\delta_{(\pi(s),\pi^*(s))}$ the *difference in reaction* to the state s.

For any policy π let d_π be its occupancy of the state only. This immediately lends to two optimisation tasks for imitation learning where the difference in reaction is *weighted* by the occupancy of either the learner or the expert:

$$\hat{\pi} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi^*}}[L(s, \pi)] \tag{3.71}$$

$$\hat{\pi} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim d_{\pi}}[L(s, \pi)] \tag{3.72}$$

The first equation (3.71) is a straightforward supervised learning problem that can be solved as such, since π^* is fixed and given, so sampling d_{π^*} is trivial. The optimising policy however, has been shown to perform up to $t^2_{\text{term}}\mathbb{E}_{s \sim d_{\pi^*}}[L(s, \pi)]$ worse than the expert policy, where t_{term} is the time horizon of the task. The behavioural drift of π from π^* may accumulate errors over time quadratically.

The authors of DAgger [RGB11] consequently pose the imitation learning task as finding the optimising policy of the second equation (3.72), that is, a policy that behaves like the expert weighted by the states it itself encounters, not by the states the expert encounters. DAgger tackles this task by implementing supervised learning on a dataset of state-action pairs $(s, \pi^*(s))$, where the states are iteratively collected as the learners experience from previous iterations. This then leads to an error that scales with t_{term} instead of t^2_{term}.

DAgger exemplifies the remarkable influence of the support of the state distribution, but also the difficulty of defining closeness of policies. Particularly interesting is that the analysis uses, as the ultimate ground truth to definitively distinguish one policy from another, the performance of the expert and the learner, which itself is a flawed metric.

3.2.2 Trust Region Reinforcement Learning Methods

Trust Region Reinforcement Learning [SLA+15] is both a concrete reinforcement learning algorithm (then referred to as TRPO) and a general approach to stabilise reinforcement

learning that has inspired others. It is based on the idea of restricting the behavioural difference between the current policy π and the next iteration of the policy $\tilde{\pi}$ that are developed sequentially during the learning process. The starting point of TRPO is to define $\eta(\tilde{\pi})$, the performance of the next policy $\tilde{\pi}$ as its advantage over the performance current policy $\eta(\pi)$, through

$$\eta(\tilde{\pi}) = \eta(\pi) + \frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim d_{\tilde{\pi}}}\left[A^{\pi}(s,a)\right] \tag{3.73}$$

where $d_{\tilde{\pi}}$ is the occupancy of $\tilde{\pi}$ and A^{π} is the advantage function of π (see Theorem 7).

TRPO uses a target function for optimisation that is an approximation in two aspects. For one, the occupancy measure of any policy is assumed to be unknown and sampled from experience. But more importantly, at any point during the optimisation, it is still unclear what the next iteration of the policy will be. That means, the occupancy $d_{\tilde{\pi}}$ of the target policy of the next iteration needs to be estimated. It is estimated through the average distribution of states d_{π} of the policy that is present in the current iteration. This results in a simple target function L based on Equation (3.73) that is iteratively maximised through gradient ascent:

$$L_{\theta}(\pi^{\tilde{\theta}}) = \eta(\pi^{\theta}) + \int_{s\in S} d_{\pi^{\theta}}(s) \int_{a\in A_s} \pi^{\tilde{\theta}}(a|s) A^{\pi^{\theta}}(s,a) \tag{3.74}$$

It is completely impractical to calculate the occupancy of the next policy $\tilde{\pi}$, so the simplification $d_{\tilde{\pi}} \approx d_{\pi}$ seems unavoidable, but it introduces a similar problem as seen in imitation learning with the mismatch of occupancy of the learner and the expert. For this reason, TRPO balances the update step that optimises the weights θ with its eponymous trust regions to enforce closeness between the old policy π and the new policy $\tilde{\pi}$ and consequently closeness of its distribution of states. Using $\overline{D}_{\mathrm{KL}}(\theta,\tilde{\theta}) = \mathbb{E}_{s\sim d_{\pi^{\theta}}}[D_{\mathrm{KL}}(\pi^{\theta}(\cdot,s) \parallel \pi^{\tilde{\theta}}(\cdot,s))]$, the Kullback-Leibler divergence of difference in reaction to states sampled from the current policy, the theoretical update step is

$$\begin{aligned} &\arg\max_{\tilde{\theta}} L_{\theta}(\tilde{\theta}) \\ &\text{subject to } \overline{D}_{\mathrm{KL}}(\theta,\tilde{\theta}) \leq \delta. \end{aligned} \tag{3.75}$$

In practice, the state distribution in Equation (3.74) needs to be estimated from experience. The performance of the current policy $\eta(\pi^{\theta})$ can be dropped for gradient ascent on $\tilde{\theta}$ as a constant, and $A^{\pi^{\theta}}$ can be replaced with $Q^{\pi^{\theta}}$ as $V^{\pi^{\theta}}$ contributes only as a constant. Then rewrite $\int_a \pi^{\tilde{\theta}}(a|s)A^{\pi^{\theta}}(s,a) = \mathbb{E}_{a\sim\pi^{\theta}(s)}\left[\frac{\pi^{\tilde{\theta}}(a|s)}{\pi^{\theta}(a|s)}A^{\pi^{\theta}}(s,a)\right]$ to achieve a formulation based on the distribution of state and actions of the current policy π^{θ}:

$$L_{\theta}(\tilde{\theta}) = \mathbb{E}_{(s,a)\sim d_{\pi}}\left[\frac{\pi^{\tilde{\theta}}(a|s)}{\pi^{\theta}(a|s)}A^{\pi^{\theta}}(s,a)\right] \tag{3.76}$$

The expression $\frac{\pi^{\tilde{\theta}}(a|s)}{\pi^{\theta}(a|s)}$ that was derived here is also called the *importance sampling ratio* [SB18]. Importance sampling is a widely used technique to analyse one distribution with

samples from another distribution [Put94]. Here, it addresses the problem of off-policy learning in reinforcement learning algorithms. Through a gradient step, the policy is updated to minimise the loss.

The constraint Equation (3.75) can be integrated in the target function with

$$D_{\mathrm{KL}}^{\max}(\theta, \tilde{\theta}) = \max_{s} D_{\mathrm{KL}}(\pi^{\theta}(\cdot|s) \parallel \pi^{\tilde{\theta}}(\cdot|s))) \tag{3.77}$$

$$C = 4\gamma(1-\gamma)^{-2} \max_{(s,a)} |A^{\pi^{\theta}}(s,a)|. \tag{3.78}$$

The authors show that this results in a guaranteed improvement of performance when under the update step

$$\arg\max_{\tilde{\theta}} \left[L_{\theta}(\tilde{\theta}) - C \cdot D_{\mathrm{KL}}^{\max}(\theta, \tilde{\theta}) \right]. \tag{3.79}$$

Using this theoretically sound C however, results in very small update steps and thus an algorithm that is not practical. The authors mention the idea of using an empirical value instead of C, but reject it due to the difficulty of tuning this hyperparameter to any new problem.

The TRPO update scheme inspires *Proximal Policy Optimisation* (PPO) [SWD+17], where instead of an explicit constraint, the target function represents in each state-action pair a lower bound for either the original expression as in Equation (3.76) or an expression where $\frac{\pi^{\tilde{\theta}}(a|s)}{\pi^{\theta}(a|s)}$ is clipped to $(1-\varepsilon)$ as a minimum or $(1+\varepsilon)$ as a maximum. The target function then is

$$\mathbb{E}_{(s,a)\sim d_{\pi}} \left[\min \left(\frac{\pi^{\tilde{\theta}}(a|s)}{\pi^{\theta}(a|s)} A^{\pi^{\theta}}(s,a), \operatorname{clip}\left(\frac{\pi^{\tilde{\theta}}(a|s)}{\pi^{\theta}(a|s)}, 1 \pm \varepsilon \right) A^{\pi^{\theta}}(s,a) \right) \right]. \tag{3.80}$$

This completely flattens the gradient anywhere beyond this trust region defined by the ε variable. To better understand what this trust region represents, fix a state-action pair (s,a) with $A^{\pi^{\theta}}(s,a) \neq 0$ in the support of d_{π} and $\varepsilon = 0.2$, the empirically best performing setting in the original paper. Then the contribution to the target function of $\frac{\pi^{\tilde{\theta}}(a|s)}{\pi^{\theta}(a|s)}$ is clipped to $[0.8, 1.2]$. If the advantage at (s,a) is positive, a change in reaction to state s going from θ to $\tilde{\theta}$ that increases the likelihood of choosing a by more than 20% is valued exactly as much as increasing it by 20%. If the advantage at (s,a) is negative, a change in reaction that decreases the likelihood of choosing a by more than 20% is valued exactly as much as decreasing it by 20%. This removes pressure towards drastic changes in reaction to states, but does not actively discourage them.

Concerning the analysis of closeness or distance between policies, TRPO and the derived PPO make the problematic assumption that the average distribution of states of the next policy can be estimated through the average distribution of states of the current policy. The algorithms address this problem with a trust region approach that keeps the current and the next policy close to each other to realise this assumption. This closeness of policies is calculated in both algorithms by a difference in reaction to states sampled from the original policy.

3.2.3 Constraint Policy Optimisation

Constraint policy optimisation [AHT+17] is an approach to learning a policy that adheres to constraints, even during training. This may be a requirement for safety, but can also be understood as a secondary, binary objective. Following the framework of a *constrained Markov decision process* [Alt99], the problem statement is changed. In addition to the reward dynamic, a number of cost dynamics C_i for $i = 1, \dots, n$ are introduced. They are defined analogously to the reward dynamics and can be understood as functions

$$C_i : S \times A \times S \to \mathbb{R} \tag{3.81}$$

$$C_i(s_t, a_t, s_{t+1}) = c_{i,t} \,. \tag{3.82}$$

This allows a formulation of the expected discounted accumulated cost $J_{C_i}(\pi)$ given a policy π:

$$J_{C_i}(\pi) = \mathbb{E}_{\tau \sim \pi} \left[\sum_t \gamma^t c_{i,t} \right] \tag{3.83}$$

For each cost function, a limit $d_i, i = 1, \dots, n$ is added. This reduces the admissible search space for policies to

$$\Pi_C = \left\{ \pi \in \Pi \mid J_{C_i}(\pi) \leq d_i, i = 1, \dots, n \right\} \tag{3.84}$$

from which the constraint reinforcement learning problem is derived as searching for $\pi^* = \arg\max_{\pi \in \Pi_C} \eta(\pi)$. As the cost of the constraints is formulated just as the rewards, analogous to the advantage function, a cost-advantage function $A^{\pi}_{C_i}(s, a)$ can be defined as the difference between one, the discounted accumulated cost given starting state s and picking the next action with the policy and two, picking the next action as a:

$$A^{\pi}_{C_i}(s, a) = \mathbb{E}_{\tau \sim \pi, s_0 = s} \left[\sum_t \gamma^t c^i_t \right] - \mathbb{E}_{\tau \sim \pi, s_0 = s, a_0 = a} \left[\sum_t \gamma^t c^i_t \right] \tag{3.85}$$

From there, the update steps in the manner of a trust region update following Equation (3.75) are defined for a practicable algorithm.

3.2.4 Novelty and Diversity

A very different application for behavioural distance is found in the concepts around *novelty* and *diversity*. These concepts are linked with evolutionary computation, a field with optimisation heuristics that work with populations of candidate solutions. That is different to the usual optimisation algorithms in reinforcement learning that are based on gradient descent and iteratively work with one candidate solution that is repeatedly improved upon. A more concrete overview of the optimisation procedures will be given in Chapter 5. Novelty and diversity are expected to improve these algorithms by fostering exploration of ever new ideas in an open-ended fashion. This is notably distinct from the focus of most optimisation algorithms that assume the existence of some optimum that should be reached as efficiently as possible.

Diversity in reinforcement learning problems is mostly viewed through the lens of a *behavioural descriptor* of an agent. Given a Markov decision process and a policy for it, the

behavioural descriptor is defined as a function that takes a sample of the policy's behaviour as input and outputs a (typically lower-dimensional) representation. Any given behavioural descriptor always implies a certain, fixed way to sample the behaviour of a policy in the given Markov decision process, but usually this sampling process is not included when referring to the behaviour descriptor:

Definition 18 (Behavioural Descriptor)**.** *Given a Markov decision process $\mathcal{M}$ with the space of episodes $\mathcal{T}$ and a policy π, a behavioural descriptor ϕ is a function that operates on the space of episodes*

$$\phi : \mathcal{T} \to \Phi. \tag{3.86}$$

The codomain of a behavioural descriptor Φ is the associated behaviour space*.*

Abusing notation slightly with $\tau \sim \pi$, a behavioural descriptor ϕ can also be understood as a random element in the behaviour space as $\phi(\pi)$.

The value $\phi(\tau)$ is the behavioural descriptor of the episode *τ, the sample $\phi(\pi)$ is a* behavioural descriptor of the policy *π.*

First introduced as part of the novelty search algorithm [LS08], the concept has been called *(behavioural) descriptor (BD)* [MC15] [VCM18], *feature descriptor* [MC15], or *behaviour characterisation/characteristic (BC)* [PSS16b] [FTN+20].

The process of applying a behaviour descriptor to a policy may drop information, especially if it transforms the observed behaviour to a lower-dimensional representation, which it typically does. At best, it may focus attention on the differences that matter, discarding noise and distractions. At worst, all that is left are incidental and irrelevant distinctions. What information is irrelevant or not, that is a distinction that is typically problem-specific and designed. Therefore, a large amount of work in distinguishing behaviour is directed towards differentiating policies along predefined axes that themselves carry relevance. Some examples of handcrafted, problem-specific behaviour characteristics include the following:

- The percentage of time of a positive roll, pitch, and yaw of the body of a robot [CCT+15].
- The proportion of time a specific leg of a walking robot touches the ground [CCT+15].
- The projection to a lower-dimensional space based on principal component analysis [CCT+15].
- The final physical position of a walker in a maze [LS08; PSS+15], or of a walking robot avoiding an obstacle [CMS+18; CD18], or of the arm of a gripping robot [CD18].
- Snapshots of the episode of positions of a walker in a maze [LS11b; PSS+15], or of the centre of mass of a walking robot [CMS+18; LS11b].
- The episode of RAM states of an Atari game [CMS+18].

Behavioural descriptors are most useful when specifically designed to differentiate policies by their most relevant properties. But there are also efforts to automate this process of finding these most relevant properties in an unsupervised manner [CTC25; GC22a; GC22b; Pao21]. The concept of behavioural descriptors is intimately linked to the endeavour of differentiating reinforcement learning policies. On the one hand, it is tempting to use these

practical ideas as a baseline when comparing policies. On the other hand, a successful approach to differentiating policies should also be applicable to these algorithms, which require a means of doing just that.

3.3 Behaviour Defines Identity

This chapter examined the critical role that behaviour and the behavioural description of policies play in reinforcement learning. Observing and learning from behaviour and its consequences is the key ingredient to reinforcement learning. This concept is already present in the statement of the framework of reinforcement learning, and its influence pervades reinforcement learning literature from the derivation of the theory to the development of a workable algorithm.

The functional view of policies is permissible if all states or state-action pairs have the same importance. In most applications involving both starting states and potentially unreachable states, the viewpoint needs to shift. The statement of the optimisation problem shifts from global optimality to local optimality, which follows optimal performance under starting states. This is the basis for policy gradient methods examined in Section 3.1.

This shift towards optimal performance under starting states is, of course, known, but it is often only acknowledged in passing if at all. After examining this property of reinforcement learning in detail, a sweeping statement can be made: *It is not just convenient to identify policies by their behaviour. It is correct.* The observable behaviour of a policy is a better representation of the results of the learning process than the function that maps the state space to the action space. After all, from the observed behaviour, the function can be reconstructed on all states except unreachable states (see Theorem 5), and the observed behaviour contains information about which situations the policy actually navigates. This information is indispensable for the policy gradients approach, which is fundamental to many reinforcement learning applications.

With this in mind, the task of understanding the space of policies and differentiating policies becomes the task of understanding the space of behaviours and differentiating behaviours. Indeed, Section 3.2 shows where behavioural characterisation and distance in reinforcement learning policies have secondary applications. Their relevance in imitation learning is obvious from the problem statement, as the expert can only be accessed through their observed behaviour. The analysis of DAgger shows an interesting detail: The error introduced by unmitigated off-policy learning compounds quadratically over the episode's length. The same problematic condition is also present in policy gradient approaches, considering the behavioural difference during optimisation of the current iteration of the policy and the next, whose behaviour is still unknown, meaning the off-policy learning problem cannot be offset, for example, with importance sampling. The use of trust regions can theoretically avoid excessive gradient steps, bounding the change of behaviour in the Kullback-Leibler divergence. The same approach can also be extended to accommodate constraints. Finally, an interest in novelty and diversity necessitates a quick and concrete distinction of policies. That leads to the straightforward behavioural distance derived from behavioural descriptors.

The next chapter on visualisation will start with this concept of behavioural descriptors and build on the more general ideas of behavioural characterisation introduced in this chapter. This work applies these new ideas to a diversity-based algorithm in Chapter 6, but they may also find fruitful integration into imitation learning, trust region optimisation, or constraint optimisation due to the relevance of behavioural characterisation to these techniques.

CHAPTER 4

Discriminating Behaviour

In their paper "Robots that can adapt like animals" [CCT+15], the authors demonstrate the capabilities of their MAP-Elites algorithm to create a repertoire of diverse yet high-performing behaviours for the same problem. In that paper, they specifically look at a simulation and later an actual physical implementation of a six-legged spider-like robot, the hexapod. As a behaviour descriptor to differentiate one strategy from another, they use the percentage of time each of the six legs touches the ground. That is a six-dimensional vector in $[0, 1]$ where the value of each entry encodes the ratio of one of the six legs, such that 0 means that this leg never touches the ground and 1 means that this leg touches the ground for the whole simulation. They found that this behaviour descriptor manages to foster a population of diverse gaits, a statement that is validated by the idea of damage adaptation. By damaging a leg or a joint, the best gait when undamaged may have a severely worse performance. But in the repertoire, they were still able to find gaits that were very successful despite the damage. There is, however, an inherent dependence on the behaviour descriptor "touching the ground with a certain leg" and the damage to a certain leg. For example, a gait that does not use a certain leg is probably resistant to damage to that leg. So even though there is an intuitive connection between creating diverse solutions to a problem and the robustness of this population of solutions against changes from the problem setup, it is not immediately clear that this is an inherent property of diversity. It may well be a property that is induced by the specific choice of behaviour descriptor that is chosen specifically to counteract certain damage.

As an addendum to the paper, one author also published a video of different gaits of the hexapod [Mou15]. This video appeals to the human understanding of different behaviours. The different gaits that are found are visibly different. There is the extreme example of a gait where the robot flips on its back and moves on its knees instead of its feet. Some deeper form of diversity that transcends the choice of behaviour descriptor has been induced. It is a qualitative difference that is evident to an observer, even if not at all clear how to quantify. With this understanding, it seems possible to use a specific behaviour descriptor to differentiate a number of behaviours that are diverse not only in relation to this specific behavioural descriptor but are also diverse in a more general sense.

Continued research on this topic has recognised the limitation of having to define a specific behaviour descriptor dependent on the problem. The research on tackling this limitation has mostly been restricted to learning a specific behaviour descriptor with means of dimensionality reduction [GC22a; GC22b; Pao21; PLC+20]. The task of finding a general behavioural descriptor seems both daunting and even futile. What is even different behaviour in a general sense? It is easy to define different behaviours given a certain marker. That is a behaviour descriptor. But what would a general quantification of this abstract and vague concept of

different behaviours entail? Can this human idea of seemingly different behaviour even be meaningfully quantified?

The inherent noisiness of all reinforcement learning problems provides a first clue to address this problem. It is well known that reinforcement learning requires some noise for proper exploration [SB18]. This is even the case in those benchmark problems that are not noisy at all, like the CartPole environment, where some randomness is introduced by taking the starting state randomly out of a set of eligible states [BCP+16]. It also manifests in learning algorithms that policies take probabilistic decisions either as part of the standard behaviour or as an aberration to their standard behaviour, specifically to improve the learning process [HZA+18; SWD+17].

Given that noisiness, the same policy rolled out in the same environment two times may produce two very different episodes. However, the same policy given the same environment with only marginally different random noise should generally exhibit similar behaviour. At the very least, given a way to discriminate between two sets of observed behaviour, the behaviour of one agent should be closer to other instances of its own behaviour than when comparing it to the behaviour of another agent. Or, to make it more specific: Can you recognise a robot by its gait?

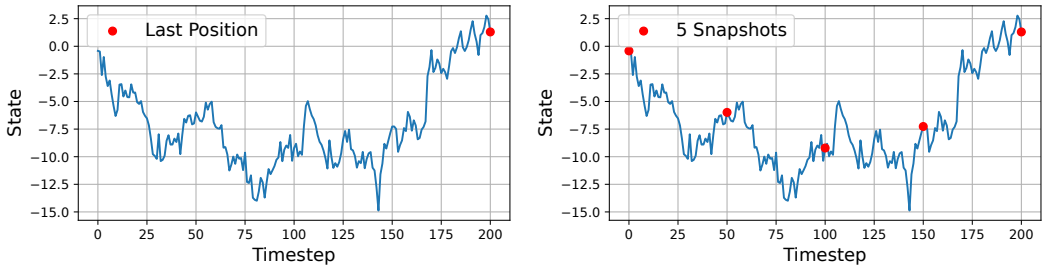
To answer this question, in Section 4.1 a selection of behavioural distances is defined that is based on the considerations of the last chapter. In Section 4.2, the behavioural distances are checked on a data set where the difference between policies and episodes is unclear. Empirically, the importance of sampling multiple episodes to capture the behaviour of a policy is shown. Not only is the influence of random initialisation during learning crucial for the outcome of the learning process, but the random initialisation of an episode is also crucial to the actual behaviour observed. These random fluctuations can be filtered out to improve the accuracy of the behavioural distance. These ideas are then put to the test, first by evaluating the data itself in Section 4.3, and second by applying them in other contexts in Section 4.4.

4.1 Behavioural Distance Measures

The analysis of the behavioural characterisation of policies in Section 3.1 introduces two pathways that may lead to a general distinction of behavioural data. One, there is the characterisation of policies as the random element of episodes in the environment. Second, derived from that, is the occupancy, a distribution in the state-action space. Both behavioural characterisations can inspire a definition of behavioural distances. That requires either a metric on time series or a metric on sampled distributions.

4.1.1 Behavioural Descriptors

There are potentially endless ways to describe behaviour in a custom problem-specific way. Understanding the behaviour of policies through the lens of such behaviour descriptors amounts to sampling a random element that is derived from the episodic characterisation of policies. These behaviour descriptors typically drop information, i.e., the original behaviour cannot be reconstructed from the description. A typical shortcut that is found in novelty- or diversity-based evolutionary algorithms is taking the last state of the episode [GC22b; MC15; PLC+20], or for a more general description of what happened a series of snapshots of states encountered at fixed steps in the series for example after 20, 40, 60, 80, and 100% completion [GC22b; VCM18], see Figure 4.1.



(a) Last position **(b)** 5 snapshots

Figure 4.1: This sketch depicts simple behavioural descriptors with a hypothetical 1-dimensional state space. Left the last position describes the whole time series, right 5 snapshots are taken to describe it.

When opting for the last state of an episode, it may be the complete state, but also just selected parts of the state, like the x-y position of a walker in a maze [MC15] or a lower-dimensional representation of the state [GC22b; PLC+20]. The special position granted to the last state is also motivated by maze-like problems, where the last state encodes important information for the solution of the task [PSS16b], but it can also be interpreted as the culmination of all past behaviour of an episode, which is therefore also representative of all past behaviour.

Using snapshots of the episode is another obvious idea: It casts episodes of potentially different length into the same space and significantly reduces the size of the data, but still maintains a good amount of the original observations. However, it still may suffer from the curse of dimensionality [AHK01; Bel57]. Each snapshot of the episode increases the dimensionality of the behavioural descriptor linearly, which creates a very large behaviour space quickly. Also, the informational value of the Euclidean distance suffers in higher dimensions [AHK01].

Definition 19. *Let $\tau = (s_i, a_i)$ be an episode of finite length. Define the two naive unsupervised behavioural descriptors, ϕ_{ls} as the* last state behavioural descriptor *and $\phi_{snap\ K}$ as the* K snapshot behavioural descriptor*:*

$$\phi_{ls}(\tau) = s_{|\tau|} \tag{4.1}$$

$$\phi_{snap\ K}(\tau) = \left(s_{\lfloor (j|\tau|+1)/(K-1) \rfloor}\right)_{j=0,\dots,(K-1)} \tag{4.2}$$

The operator $\lfloor x \rfloor$ signifies rounding down to the next integer value.

4.1.2 Time Series Classification

Episodes, as a way to understand the behaviour of a policy, lead to time series as the basis of data. It makes sense to reference how time series are compared in other contexts. *Time series classification* [BLB+17; RFL+21] is the study of categorising a given set of time series. Typically, this concerns classification in the sense of a supervised task, where some data is already labeled as belonging to a class. From this, a model is built, which in turn can categorise unlabeled data accordingly. Depending on the setup, some of these methods can also *cluster* the data, the unsupervised equivalent to classification where data is categorised without any labels to begin with.

Assuming a metric on the behavioural data of a reinforcement learning agent distinguishes that behaviour in a meaningful way, this metric should be at least somewhat successful in classifying or clustering episode data. That means, based on the observed data, it should be possible to create clusters that relate to some underlying metadata or build an accurate classification model. Given a reinforcement learning problem, this metadata could be the choice of algorithm that trains a policy, the architecture of the policy function, and the hyperparameters that are used for training. But most immediately, it is to differentiate policies based on their behaviour, that is, being able to cluster instances of observed behaviour as similar if they came from the same source, the same agent.

4.1.3 Dynamic Time Warping

While there are multiple, very intricate ways to classify time series, a surprisingly strong and simple baseline in contemporary time series classification is *dynamic time warping* (DTW) to calculate distances between pairs of time series combined with a distance-based classifier like a k-nearest-neighbour approach [RFL+21]. Dynamic time warping can credibly find meaningful differences in time series data. In contrast to most other contemporary methods, dynamic time warping does not learn a model, but rather works with pairwise distances. This fits well in the task of comparing reinforcement learning agents based on behaviour, where behaviour can be observed repeatedly on demand and is not static. It can therefore serve as a baseline and an initial validation method for any more abstract or general distance measure on the behavioural data of reinforcement learning agents.

Dynamic time warping is a technique to reparametrise time in two discrete time series in a way that the reparametrised series match as well as possible. Formally, given two time series $X = (x_i)_{i=1,\dots,n_x}$ and $Y = (y_i)_{i=1,\dots,n_y}$, call a *warp path* an increasing series of indices $(i_1, j_1)_{k=1,\dots,n}$ that starts with 1, so $i_1, j_1 = 1$, that ends with the maximum index of either path, so $i_n = n_x$, $j_n = n_y$, and that increases by either 0 or 1 at every time step, $i_{k+1} \in \{i_k, i_k + 1\}$, $j_{k+1} \in \{j_k, j_k + 1\}$. Given a distance measure d that acts on the elements of the time series as $d(x_i, y_j)$, the warp distance for a path is defined as the sum of distances of points in X and Y along the warp path. Dynamic time warping finds the optimal path that minimises this distance. The DTW distance of that minimising path is then a distance measure between two time series:

$$\mathrm{DTW}_d(X, Y) = \min_{\text{warp path } (i_k, j_k)} \sum_{k=1}^{n} d(x_{i_k}, y_{j_k}). \tag{4.3}$$

This gives a distance that, as well as possible, ignores differences in the time series that arise from a mismatch of time parametrisation. When comparing two time series, it is often not essential when precisely a certain pattern occurs, but rather if it occurs and how often. Take, for example, a time series that captures the movement of a robot: It may be essential information that a certain leg moves up and down with a certain frequency, but the precise moment the movement starts is not. Figure 4.2 shows a graphical explanation of dynamic time warping where the L_1 norm takes the role of the distance measure with $d(x, y) = \|x - y\|_1$. The definition of Equation (4.3) naturally encompasses multi-dimensional time series too, even time series in non-euclidean spaces, as long as a metric is defined.

As a practical note, in any test presented in this work, not a full DTW but rather FastDTW [SC07] is used. While DTW is fast for time series classification standards, it is somewhat

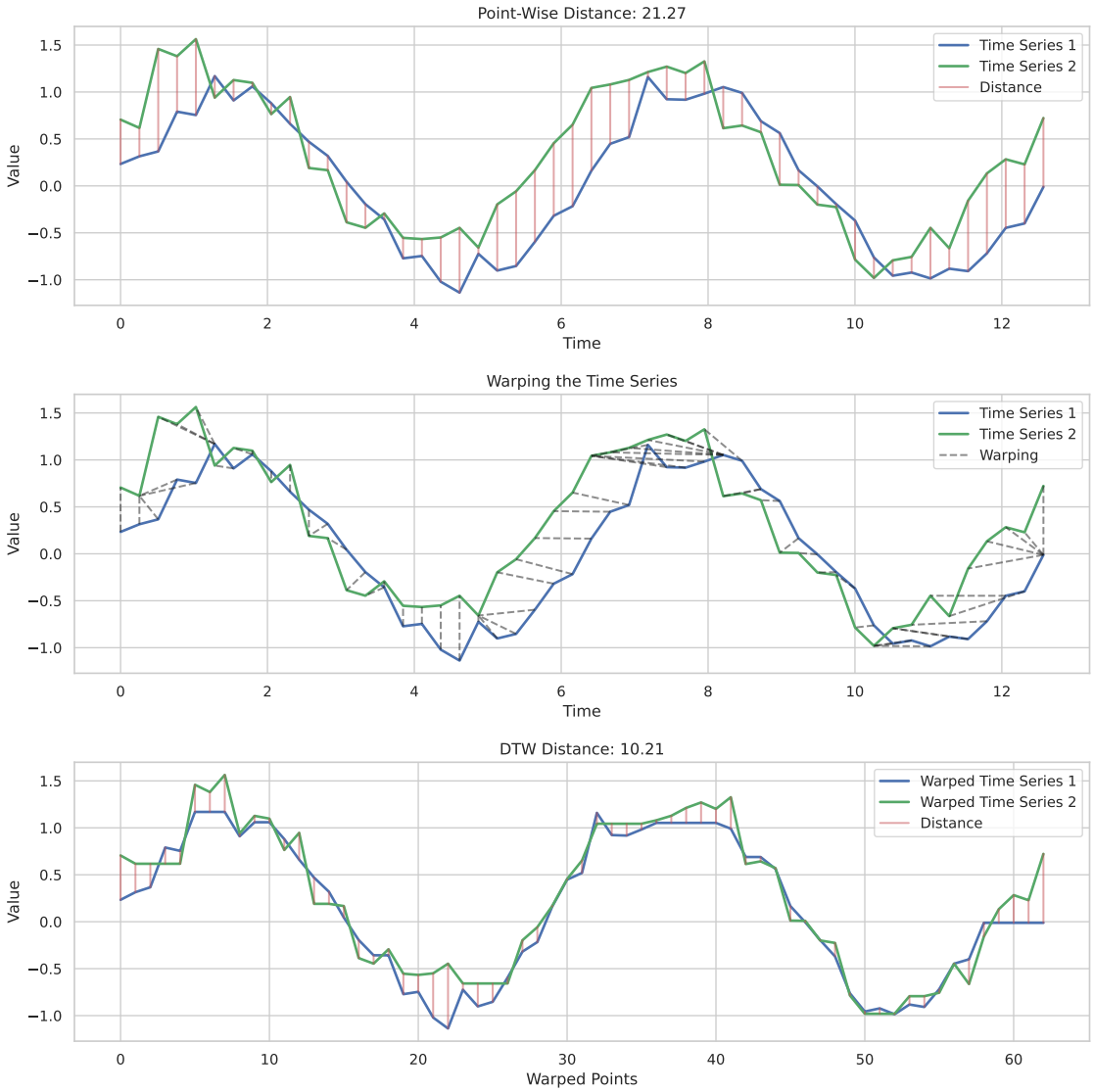


Figure 4.2: A visual comparison of pointwise distance and time warped distance of two synthetic 1-dimensional time series. The first plot shows a pointwise distance measure, the second plot shows the matching of points using dynamic time warping with an L_1 norm. The third plot shows the warped time series, from which a pointwise difference is calculated.

slow in comparison to the usual ideas to compare reinforcement learning agents based on their behaviour. FastDTW offers a significant speed up with the trade-off that points from one time series cannot be matched with points from the other time series that are arbitrarily far away. In the context of reinforcement learning agents, this may actually be an upside, as episodes typically start in a very similar manner. That means a more local time warping may capture the specific nature of this data even better.

4.1.4 Signature Transform

The *signature* of a path is defined as a real-valued vector which describes properties of the path [CK26]. The values of this vector can be understood as a generalisation of statistical moments. In its pure form, it is described on a path $X : [a, b] \to \mathbb{R}^d$. From the perspective of reinforcement learning, an episode in an environment can be understood as the discretisation of such a path, or even more generally, as a full description of a piecewise linear path.

Given two one-dimensional paths $X, Y : [a, b] \to \mathbb{R}$, define the *integral of Y against X* as

$$\int_a^b Y(t)X'(t)\,dt. \tag{4.4}$$

For a multi-dimensional path $X : [a, b] \to \mathbb{R}^d$ with $X(t) = (X_i(t))_{i=1,\dots,d}$, first define a new path on $t \in [a, b]$ as the integral of $\mathbb{1}$ against X_i from a to t.

$$S(X)_a^i(t) = \int_a^t X_i'(s)ds = X^i(t) - X^i(a) \tag{4.5}$$

This path can now be integrated against another component X_j up to $t' > t$ to create another path, and so on. With this idea, the *k-fold* integral along the indices $i_1, \dots, i_k \in \{1, \dots, d\}$ can be defined recursively:

$$S(X)_a^{i_1,\dots,i_k}(t) = \int_a^t S(X)_a^{i_1,\dots,i_{k-1}}(s)(X^{i_k})'(s)\,ds \tag{4.6}$$

$$= \int_a^t \int_a^{t_k} \dots \int_a^{t_2} (X^{i_1})'(t_1) \cdots (X^{i_k})'(t_k)dt_1 \cdots dt_{k-1}\,dt_k \tag{4.7}$$

From this definition, the real valued signature coefficient $S(X)^{i_1,\dots,i_k} \in \mathbb{R}$ is set as the value $S(X)_a^{i_1,\dots,i_k}(b)$.

Definition 20. *For a multi-dimensional path $X : [a, b] \to \mathbb{R}^d$, call the sequence defined by*

$$S(X) = (1, S^1(X), \dots, S^d(X), S^{1,1}(X), S^{1,2}(X), \dots, S^{d,d}(X), S^{1,1,1}(X), \dots) \tag{4.8}$$

the signature of X. Call $S^k(X)$ the signature of X of depth k, a vector that contains every entry in $S(X)$ that is l-fold integral with $l \leq k$.

From this definition, already one interesting quality of the signature is evident: It transforms time series of any finite length into a vector whose length is only defined by the depth of the transformation and the dimension of the state(-action) space. The signature of depth k on a d-dimensional path is precisely of dimension $\sum_{i=0}^k d^i$: $S^k(X) \in \mathbb{R}^{\left(\sum_{i=0}^k d^i\right)}$

The signature transform is hailed as an "excellent candidate as a feature extractor of time series". The most important piece of evidence for this claim is that the signature transform is a bijective operation on compact paths. It can even be shown that this also holds for the random processes that create compact paths and their expected signatures [LNS+24].

Fixing an environment and a policy π, this is exactly the setup for differentiating these policies. Rolling out episodes $\tau \sim \pi$ can be interpreted as a random process. The episodes are compact as long as every episode terminates in finite time and the state(-action) space is

bounded. The expected value of the signature of a policy can be estimated by the signature of a certain depth of one sampled episode or even as the average of multiple sampled episodes.

The signature transform has also successfully been used in time series classification [MFK+20], but has some significant advantages in comparison to dynamic time warping for practical applications when working with reinforcement learning problems: It serves as a behavioural descriptor in the sense that it is a function of behaviour first, and the image of that function is equipped with a metric. The majority of the computational effort lies in calculating this behavioural descriptor and not in calculating the metric.

4.1.5 Optimal Transport

With the occupancy measure, behavioural data can be understood as the density of the visitation frequency in the state-action space. A direct way to compare samples of unknown distributions is with the idea of *optimal transport*. The concept and name are inspired by operations research [HL01]:

Definition 21 (Optimal transport problem)**.** *Let* $(s_i)_{i=1,\dots,N} \in \mathbb{R}$ *be the supply,* $(d_j)_{j=1,\dots,M} \in \mathbb{R}$ *the demand and* $(c_{ij})_{i=1,\dots,N,j=1,\dots,M} \in \mathbb{R}$ *the cost of transport between the supply location* i *and the demand location* j*. Let supply and demand fulfil the* feasible solutions property $\sum_{i=1,\dots,N} s_i = \sum_{j=i,\dots,M} d_j$*. The* optimal transport problem *is to minimise the* transport cost

$$Z = \sum_{i=1,\dots,N} \sum_{i=j,\dots,M} c_{ij} x_{ij} \tag{4.9}$$

under the conditions

$$\sum_{i=1}^{N} x_{ij} s_i = d_j \qquad \text{for } j = 1, \dots, M, \tag{4.10}$$

$$\sum_{j=1}^{M} x_{ij} d_j = s_i \qquad \text{for } i = 1, \dots, N, \tag{4.11}$$

$$x_{ij} \geq 0. \tag{4.12}$$

The solution Z *to the optimal transport problem is the* optimal transport cost*.*

Definition 22 (Optimal transport distance)**.** *Let* $(x_i)_{i=1,\dots,N}, (y_j)_{j=1,\dots,M} \in \mathbb{R}^n$ *be samples of two unknown distributions. Set supply* $(s_i)_{i=1,\dots,N} \equiv \frac{1}{N}$*, demand* $(d_j)_{j=1,\dots,M} \equiv \frac{1}{M}$ *and transport cost* $c_{ij} = \|x_i - y_j\|$*. The optimal transport cost of the problem is the* optimal transport distance *between the sampled distributions.*

As with dynamic time warping, most of the computational effort lies in calculating the pairwise distance, not in creating a behavioural descriptor. It can be shown that minimising the transport cost scales with the dimension of the cost matrix d as $O(d^3 \log d)$, and that the optimal transport distance, as defined above, is actually a distance [Vil09]. An implementation from the POT [FCG+21] library was used for all the following practical applications.

4.2 Visualisation of CartPole Policies

These ways to differentiate behaviour can now be tested against the first assumption that one instance of a policy's behaviour should be closer to another instance of that same policy's

behaviour than to an instance of another policy. The analysis of different techniques is first conducted in the CartPole environment and later verified with data from the Ant environment, see Subsection 2.4.1.

4.2.1 Data Creation

Using alternately different reinforcement learning strategies and different seeds for random number generation during learning, it is easy to find parameterisations of small deep neural networks that act as agents in the CartPole environment. Usually, an agent that can confidently survive an episode of 500 time steps can be considered a proper solution to the problem. Given some solutions to the problem, they can be rolled out several times to create a dataset of the behaviour of the solutions. Now, already the easiest discriminator of different behaviour, that is, the success of a certain agent, can no longer distinguish between these solutions, as they all survive 500 timesteps to incur a reward of 500.

The agents for the CartPole data are created by fixing a reinforcement learning algorithm out of ARS, PPO, TRPO, and A2C, and three different seeds for random number generation during learning. This includes sampling of random noise, for example, for ARS (see Subsection 2.4.2), but also to select the starting state s_0 for every new episode created. All algorithms except ARS find weights for a fully connected feed-forward neural network with ReLU activation functions. ARS searches for weights for a linear transformation from the state to the action space. In total, 12 policies are trained, three policies for each of the four methods. Training continues until the policy solves the problem 100 out of 100 times. That means that some policies have had more data to train, others more computing power, but from all of them, a set of 100 episodes with maximal reward is collected.

These 100 episodes serve as the data base of the analysis. For every policy, they are created using the same 100 seeds for random number generation to initialise the episode and to determine the stochastic behaviour of the policy if applicable. Episodes are initialised values sampled uniformly between ± 0.05, so $s_0 = (s_{0,i})$ and $s_{0,i} \sim \mathcal{U}_{[-0.05,0.05]}$ for $i = 1,2,3,4$. With a state space $S \subset \mathbb{R}^4$, action space $A = \{\pm 1\}$ and 500 time steps per episode, the observed data for 12 agents with 100 episodes has dimension $(12 \times 100 \times 500 \times 5) = 1{,}250{,}000$.

In conjunction with a metric on this behavioural data, the distance between one policy and another can be empirically calculated:

Sample for policies $\pi^{(1)}, \pi^{(2)}$ episodes $\tau^{(1)} \sim \pi^{(1)}$ and $\tau^{(2)} \sim \pi^{(2)}$ to estimate a distance between policies with

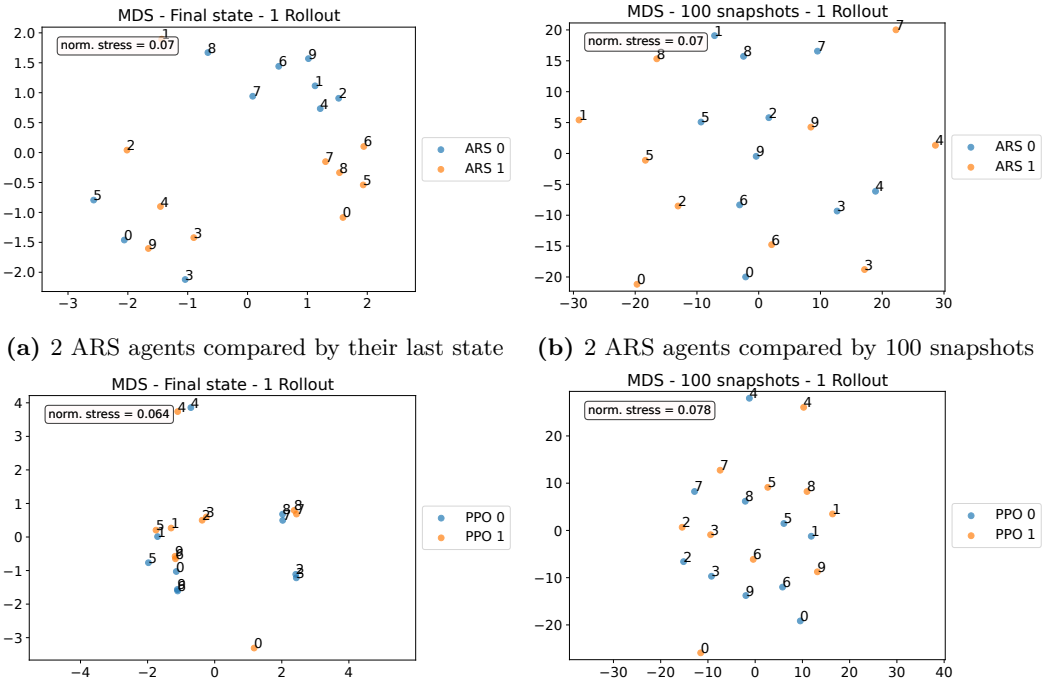
$$d(\pi^{(1)}, \pi^{(2)}) = \mathbb{E}\left[d(\tau, \tilde{\tau}) \mid \tau \sim \pi^{(1)}, \tilde{\tau} \sim \pi^{(2)}\right] \tag{4.13}$$

$$\approx d(\tau^{(1)}, \tau^{(2)}) \tag{4.14}$$

This distance measure of policies is not a proper metric any more, as it violates the identity of indiscernible $d(\pi, \pi) \neq 0$ as soon as there is any variance in sampling $\tau \sim \pi$. As a consequence, however, the distance measure in Equation (4.14) makes it possible to test not only the distance of one policy to another but also the distance of one policy to itself.

4.2.2 Visualisation of Behavioural Descriptors

To visualise measured distances between abstract entities like policies, techniques exist that search for lower-dimensional points that represent the original distance data, such that the Euclidean distances of the embedded data represent as well as possible the original distances



(a) 2 ARS agents compared by their last state

(b) 2 ARS agents compared by 100 snapshots

(c) 2 PPO agents compared by their last state

(d) 2 PPO agents compared by 100 snapshots

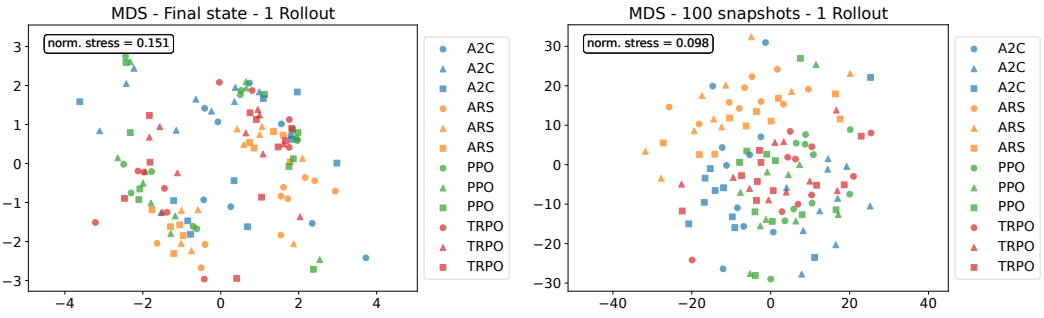
Figure 4.3: Visualisation of the behaviour of two agents trained either with ARS or PPO using either last state or 100 snapshots as behavioural descriptors.

of the original data. Depending on the use case, different ideas have been formulated about what the best representation may look like. Specifically, if it is best to retain the proper distances as well as possible, scale these distances in some way for better visualisation, or retain the order of distances as well as possible. In this work, Multidimensional Scaling (MDS, see Appendix A.1) is used.

The visualisation process here is best understood in three steps. They form a pipeline that starts with some policies and results in a visualisation of how close or different these policies are to each other and how self-similar their behaviour is.

1. Observe episodes $\{\tau_{i,j}\}_{j=0,\dots,9}$ for each policy π_i.
2. Create a distance matrix D, by applying the behavioural descriptor and measuring the distance between the behavioural description of each episode.
3. Use MDS to translate the distance matrix to 2-dimensional embedded points.

As an exploratory analysis, consider just two pairs of successful policies. One pair is trained with PPO and the other with ARS. While both methods have stochastic elements, ARS has by construction more and deeper influences of randomness than PPO. Therefore, it is reasonable to assume that the agents trained with PPO exhibit less diverse behaviour than the agents trained with ARS. To not overload the pictures, just 10 episodes per policy are selected. This way, each policy has 10 different data points associated with it. A first attempt to visualise the relationship of behaviour of these policies will be made in Figure 4.3 by comparing different episodes of two policies based on simple behavioural descriptors



(a) Last state **(b)** 100 snapshots

Figure 4.4: Visualisation of the behaviour of 12 agents trained with A2C, ARS, PPO, and TRPO using last state and 100 snapshots. Each point represents one episode, each different colour represents a different learning algorithm, and each different marker represents a different initial seed during learning.

ϕ_{ls}, and K snapshots $\phi_{\text{snap K}}$ (see Subsection 4.1.1) and the Euclidean distance. Up to two things are a priori expected to be observable: A similarity of behaviour that comes either from the same policy or the same initialisation of the episode.

Figure 4.3 shows two visualisations of the distances of different episodes of two agents, in each picture, both trained with either ARS or PPO. Each point corresponds to one episode, and the colours represent which agent created the episode. The numbering in the names of the agents (e.g. PPO 0) indicates the seed for random number generation during learning. A number next to a point in the picture indicates the seed the was used to initialise the episode. While the pictures are all somewhat chaotic, some show some clustering, a relationship between the different seeds used for initialising episodes, and the self-similarity of behaviour of the same policy.

The two policies trained with ARS visualised with the final state descriptor in Figure 4.3(a), exhibit three clusters, two of which exclusively belong to one policy, indicating some overarching similarity of the policies' behaviour even with different initialisation of the episode. The visualisation of comparing a sampled time series with 100 snapshots in Figure 4.3(b) seems to spread out the points almost evenly, except that several points with the same seed for initialising the episode are close to each other. This is even more visible in Figure 4.3(d) for 100 snapshots with the two policies trained with PPO. In Figure 4.3(c), visualising with the final state descriptor, some self-similarity is shown with some episodes created with the same policy but using a different seed for initialising the episode being very close.

Clearly, there may be some informational value to be found here, but the initialisation of the episode seems to impact the outcome of this method strongly. This also casts doubt on the validity of the previous interpretation and on the ability of this method to capture more complex relationships. Considering a larger pool of agents, as seen in Figure 4.4, the method fails to deliver any meaningful information from the data.

4.2.3 Aggregation

To mitigate the impact of the initialisation of an episode, two approaches come to mind: One, averaging or aggregating over several episodes, or two, aligning the initialisation when comparing two policies in a way that only episodes with the exact same random behaviour

are compared. The second approach is covered in Subsection 4.4.2. However, for many use cases, this approach is not viable, for example, because perfectly reproducible initial conditions are impossible in real-life applications, but also because the performant use of GPUs introduces some randomness in simulations. The design of a metric between policies using the expected value in Equation 4.13 of samples already suggests that the accumulation of more behavioural information about a policy may increase the information value of the calculated distances.

Definition 23 (Aggregation)**.** *Given a Markov decision process $\mathcal{M}$ with the space of episodes $\mathcal{T}$, a policy π, and a behaviour descriptor $\phi : \mathcal{T} \to \Phi$, call an* aggregation *of k episodes a function $\mu : \Phi^k \to \Phi$.*

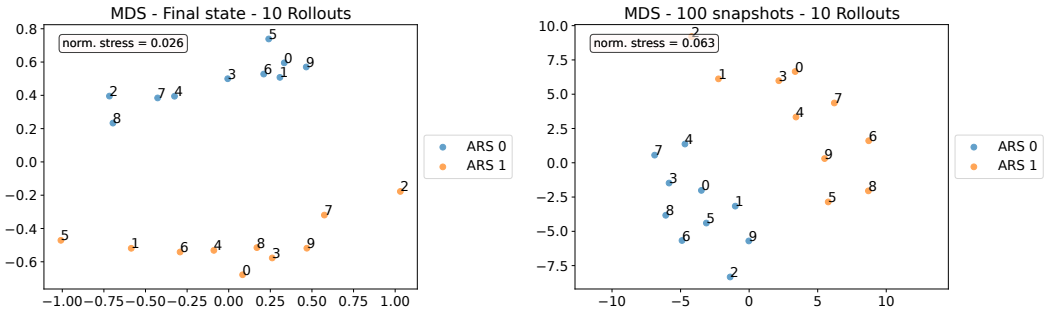
For the simple behavioural descriptors "last state" ϕ_{ls}, and "K snapshots" $\phi_{\text{snap K}}$, there is no natural way to properly aggregate multiple episodes. The makeshift approach taken here, is simply taking the mean $\overline{\phi(\tau_i)} = \frac{1}{N}\sum_{i=1}^{N} \phi(\tau_i)$ over the different values found when rolling out multiple episodes $\{\tau_i\}_{i=1,\dots,N}$.

Already, the behavioural descriptors introduced earlier do not utilise the full data from the observed behaviour. Different strategies may arrive at the same result and be grouped when considering the final state. This drawback worsens when calculating the mean of the final state. One policy may always end in one of two places, another may always end in their mean. Aggregating their final states by taking the mean will now group them, even though they were distinguishable before. Even if it is not that pronounced, taking snapshots provokes a similar effect. But this method will be able to absorb some of the random fluctuations that come with different initialisation of an episode.

To test the impact of aggregation through averaging, for each policy, the 100 episodes collected are divided into ten sets of ten. This way, there are again 10 samples of behaviour for each policy, only this time each sample contains the information of 10 episodes instead of just one. The choice of 10 episodes is somewhat arbitrary, but it offers a good trade-off between increased computation time and a visible impact, as this chapter will show. When calculating the behavioural descriptor for such a sample, the behavioural descriptor for each episode is calculated, and the 10 resulting descriptors are averaged, taking the Euclidean mean. So, in Figure 4.5, the policies are compared as before, but this time each point represents 10 episodes rolled out with one policy.

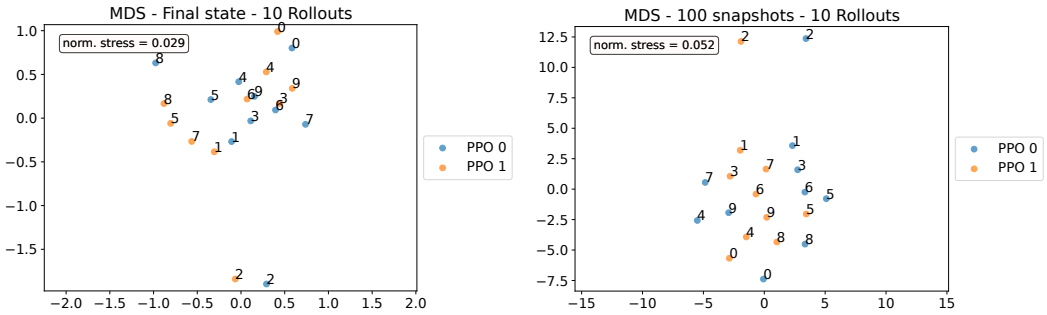
The plot of the two policies trained with ARS groups the episodes of the two agents into two distinct clusters both with last state (Figure 4.5(a)) and with 100 snapshots (Figure 4.5(b)) to describe their behaviour. This clear separation is hardly a coincidence: The two policies exhibit different behaviour, and this technique is able to pick up on the difference when filtering out some random fluctuations.

Concerning the visualisations of the distances of episodes of two agents trained by PPO (Figures 4.5(c), 4.5(d)), the training process should, by construction, be less stochastic, so the policies trained with PPO and different seeds during learning may exhibit less diverging behaviour than when comparing two policies trained with ARS and different seeds during learning. Indeed, in both plots, the embedded points are not separated but still fully intermingled. With both behavioural descriptors, most behaviour that is observed using the same initialisation when rolling out the policy is close by. This implies that both policies are actually very close in behaviour, in contrast to the two policies trained with ARS that are separated.



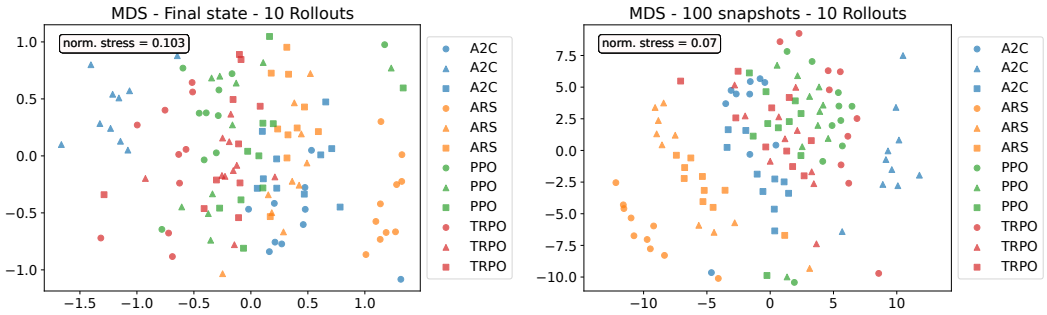
(a) ARS compared with last position

(b) ARS compared with 100 snapshots



(c) PPO compared with last position

(d) PPO compared with 100 snapshots



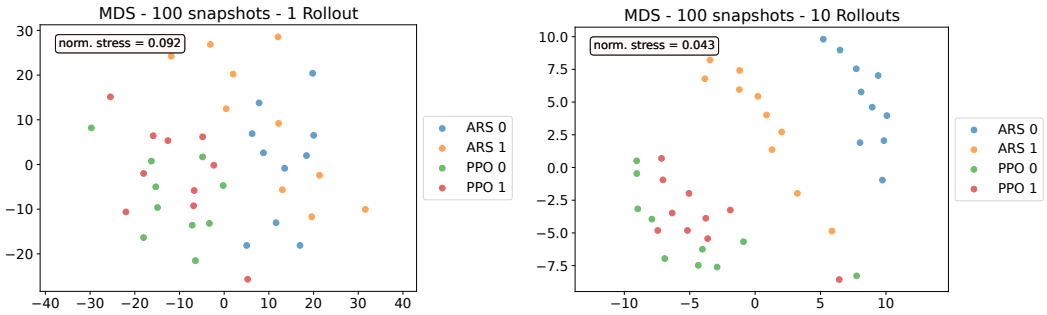
(e) All 12 policies compared with last position

(f) All 12 policies compared with 100 snapshots

Figure 4.5: Visualisation of the behaviour with MDS. Here, each point represents the mean value of the behavioural descriptor 10 episodes observed for that agent.

While this is an encouraging result for this endeavor, a broader visualisation in Figures 4.5(e) and 4.5(f) of several policies still turns out to be a mostly chaotic plot. The plot using 100 snapshots in Figure 4.5(f), however, already seems to show something more than just chaos: The three policies trained with ARS seem to be separating from the cohort, just as one A2C policy.

The impact of aggregation is clearly visible in Figure 4.6 when comparing both ARS and PPO agents. Using no aggregation leads to a picture in Figure 4.6(a) that seems to be just noise, although episodes created by policies trained by ARS are mostly separated from policies trained by PPO. The aggregation of 10 episodes in Figure 4.6(b), however, leads to a picture with two distinct clusters for the two agents trained with ARS and one cluster for the agents trained with PPO.



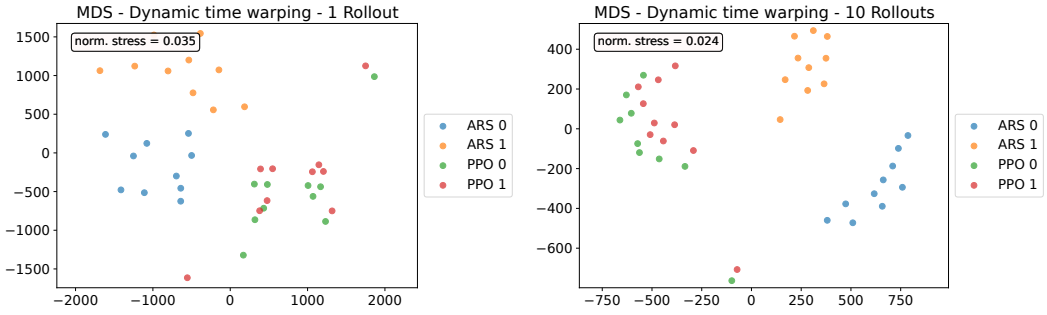
(a) No aggregation **(b)** 10 episodes aggregated

Figure 4.6: Visualisation of episodes of the four different agents trained with PPO and ARS. Distances are calculated as the Euclidean distance between 100 snapshots of episodes. Left, distances are calculated from just one episode, and right, distances are calculated after aggregating 10 episodes by taking the mean value of the respective behavioural descriptors. The visualisation in dimension two is achieved with MDS.

Aggregation, even by only averaging, has a visible effect in filtering out noise. This is a testament to how strong random initialisation influences the outcome of an episode in reinforcement learning. More sophisticated ways to aggregate are introduced with the new behavioural descriptors in Subsection 4.2.4. An argument for aggregating behaviour over averaging distances is made in Subsection 4.3.1.

4.2.4 New Behavioural Distances

When comparing full episodes or a large number of snapshots, the Euclidean distance as a measure is not ideal. Dynamic time warping as a behavioural distance measure is a natural way to compare behaviour that is understood in the episodic sense. As with the last state and when sampling snapshots, aggregation by taking the mean, while still promising, is not really theoretically sound. To aggregate episodes before applying dynamic time warping, the mean value for each time step is calculated over all 10 episodes. There is no problem here with different-length episodes by construction of the problem, but in other environments, that would also require some creative solution.



(a) No aggregation **(b)** 10 episodes aggregated

Figure 4.7: Visualisation of episodes of the four different agents trained with PPO and ARS. Distances are calculated with dynamic time warping. Left, distances are calculated from just one episode, and right, distances are calculated after aggregating 10 episodes by taking the mean of these episodes. The visualisation in dimension two is achieved with MDS.

Table 4.1: How different behavioural descriptors are aggregated.

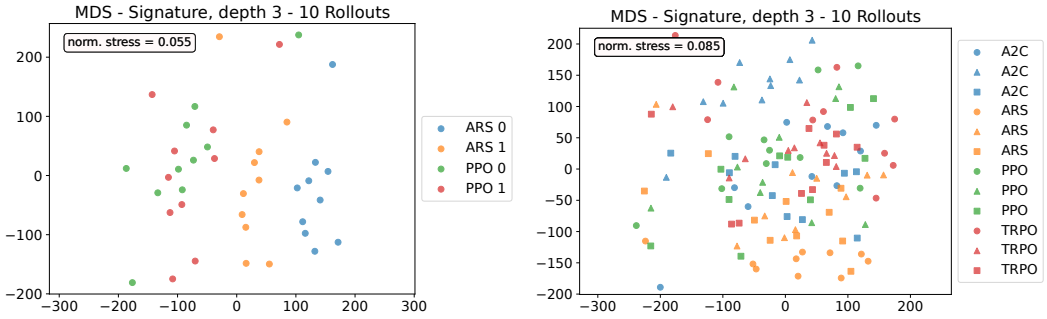
Metric	Types of aggregation
Snap. 100	Mean of Episodes
Last Pos.	Mean of Episodes
Sgnt. 3	Mean of Signatures
OT	Concatenation
DTW	Mean of Episodes

The resulting visualisation shown in Figure 4.7 suggests that dynamic time warping is a more sensitive way to distinguish behaviour. It is able to pick up the dynamics between the four agents in Figure 4.7(a) that was picked up before only after aggregating 10 episodes, but it also shows the impact of the aggregation in Figure 4.7(b): The left-hand side with no aggregation shows both policies trained with ARS distinct from each other and from the two policies trained with PPO. The policies trained with PPO are spread out over the same area, and points representing episodes created with the same seed for initialising them are close to each other. This relationship becomes more pronounced in the picture with aggregation. The three different clusters are distinctly separate now.

A major conceptual drawback of the introduced ideas so far is the inadequate way to aggregate the observation of several episodes even though that seems to be an important puzzle piece. Occupancy measure and the signature transform as behavioural descriptors both address this problem, providing a theoretically sound way to aggregate multiple episodes.

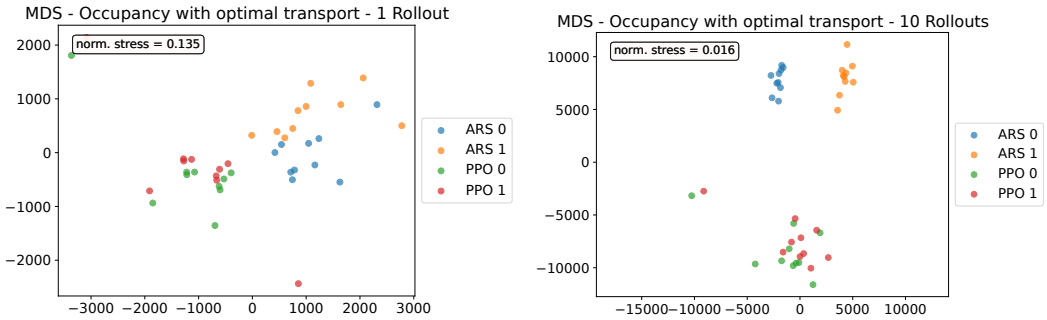
Understanding the environment and policy as a random variable that samples time series, the expected signature of that random variable is a unique representation of the random variable. The signature of any episode sampled from this random variable is an approximation of the expected signature, but probably a bad approximation. The more time series are sampled and averaged, the more they should approximate the expected signature of the random variable and therefore characterise the policy's behaviour in that environment.

For optimal transport, sound aggregation is even simpler. Each policy can be represented by any number of episodes by simply creating a list of state-action pairs encountered. The more episodes are sampled the more accurate is the estimated occupancy measure.



(a) No aggregation, 4 agents **(b)** 10 episodes aggregated, 12 agents

Figure 4.8: Visualisation of episodes with no aggregation and four agents and aggregation over 10 episodes and twelve agents, using signature transform with depth 3. Aggregation is done by taking the mean value of the signature transform of the episodes.



(a) No aggregation **(b)** 10 episodes aggregated

Figure 4.9: Visualisation of episodes with occupancy measure and optimal transport.

Summarising, for the behavioural distance based on last position, behavioural distance based on 100 snapshots, and the behavioural distance with dynamic time warping, aggregating is equivalent to averaging episodes. While averaging episodes seems to be beneficial in the scope of this analysis, it may also introduce new problems (see Subsection 4.2.3). For the behavioural distance based on signature transform and optimal transport, aggregating is different from averaging episodes. In both instances, the information of multiple episodes is combined in a way that the distance is measured more accurately. Table 4.1 gives a summary of the different types of aggregation.

Figure 4.8 shows that the signature transform is not particularly suited to comprehend the given data. Depicting just four agents is about as informative as taking 100 snapshots as a behavioural descriptor and averaging to aggregate, see Figure 4.6(b). Depicting all 12 agents, no information can be read from the plot, even though the normalised stress seems to indicate the visualisation in two dimensions is fine.

Understanding the episodes as samples from the occupancy measure, however, leads to significantly clearer pictures even without aggregating multiple episodes. In Figure 4.9, the information for four agents is as clear as before, even using just one episode as observation data in Figure 4.9(a), and very clear after aggregating 10 episodes in Figure 4.9(b).

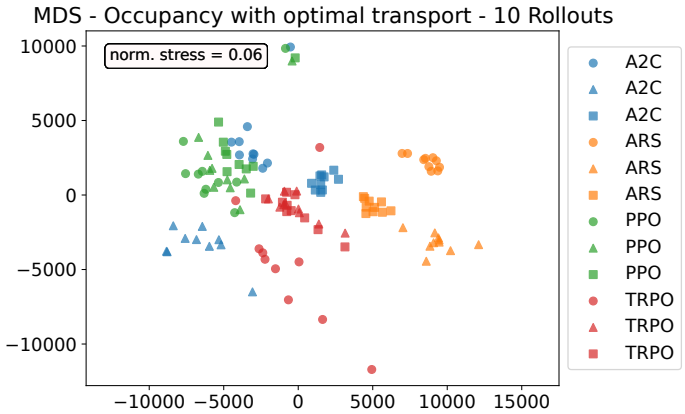


Figure 4.10: Visualisation of episodes with aggregation of 10 episodes and occupancy measure with optimal transport.

This behavioural descriptor still creates a meaningful picture of the relationship of all twelve policies trained with four different algorithms in Figure 4.10. Disregarding some outliers, it suggests a clear relationship between the policies regarding their behavioural differences: All three policies trained with ARS are separate from the cohort and from each other. Two policies created with TRPO are very similar but they differ from the third and the rest of the cohort. All three policies trained with PPO and one policy trained with A2C are very similar. The other two policies trained with A2C are distinct from the cohort.

Of course, it is difficult to really assess the success of such a method, as there is no clear ground truth. The relationship of observed behaviour to the policy it was sampled from, and the relationships of the learning algorithm to the policy, are very appealing as they corresponds with intuitive expectations. At last, this relationship seems to be resolved in Figure 4.10.

4.3 Metaevaluation of Metrics

When comparing different approaches to visualisation, a good first indicator for quality is order over chaos. If one approach can identify a preexisting pattern, in this case, the relationship between episode and policy and the relationship between policy and learning algorithm, that itself affirms the approach. After all, information is probably either destroyed or retained. It is unlikely that a pattern that suits other available information is created by chance.

To clearly demonstrate and quantify the capabilities of the presented approaches in finding these relationships, this section introduces two additional techniques that further analyse the different distance measures. First, the comparison of the average distance based on rollouts of policies to test the relationship between episodes initialised with the same random seed and episodes initialised with different random seeds. Second, a separation heuristic, which automates the evaluation of the visualisation regarding the question of whether the behaviour of one policy is visibly distinct from the behaviour of another policy.

4.3.1 Comparing Average Distances

So far, distance measures have been used for visualisation purposes. But in the process of creating a visualisation based on the calculated pairwise distance, information may get lost, or patterns may arise that are not actually supported by the data. Looking at the distance measurements directly and skipping the visualisation process, the data is less processed. While it is more effort to understand relationships from the data, it also avoids the possible interference from the visualisation process.

Given two policies π_1, π_2 and two random seeds $\omega, \tilde{\omega}$ for initialisation of episodes, and a behavioural distance measure d, a number of comparisons can be made. Episodes created under the same policy and with the same initialisation have a distance of $d(\pi(\omega), \pi(\omega)) = 0$ for all measures analysed in this chapter. Of interest are the relationships of the following distance measurements:

- $d(\pi(\omega), \pi(\tilde{\omega}))$, the distance of behaviour observed under the same policy but different random seeds for initialisation of episodes shows how spread out the observed behaviour is under the distance measure. A high value here implies a large spread in behaviour in that policy just from different initialisations of the episode.

- $d(\pi_1(\omega), \pi_2(\omega))$, the distance of behaviour observed under different policies and the same random seed for initialisation of episodes gives a true distance measurement between the policies under the behavioural distance measure d.
- $d(\pi_1(\omega), \pi_2(\tilde{\omega}))$, the distance of behaviour observed under different policies and a different random seed for initialisation of episodes gives a noisy distance measure that is more true to real-world applications, where no experiment can be repeated under exactly the same circumstances.

Empirically, these distances are calculated and averaged here with the same data as used before in the following way. For policies $\{\pi_i\}_{i=1,2}$, 10 data points $\{\pi_i(\omega_j)\}_{j=1,\dots,10}$ are collected.

- $\overline{d(\pi_1(\omega), \pi_1(\tilde{\omega}))} = \frac{1}{45} \sum_{1 \le j < k \le 10} d(\pi_1(\omega_j), \pi_1(\omega_k))$ is the average distance of $\pi_i(\omega_j)$ to $\pi_i(\omega_k)$ with $j \neq k$, an average over 45 measured distances.
- $\overline{d(\pi_1(\omega), \pi_2(\omega))} = \frac{1}{10} \sum_{j=1}^{10} d(\pi_1(\omega_j), \pi_2(\omega_j))$ is the average distance of $\pi_1(\omega_i)$ to $\pi_2(\omega_i)$, an average over 10 measured distances.
- $\overline{d(\pi_1(\omega), \pi_2(\tilde{\omega}))} = \frac{1}{45} \sum_{1 \le j < k \le 10} d(\pi_1(\omega_j), \pi_2(\omega_k))$ is the average distance of $\pi_1(\omega_j)$ to $\pi_2(\omega_k)$ with $j \neq k$, an average of 45 measured distances.

That means that

$$\overline{d(\pi_1(\omega), \pi_2(\omega))} = \overline{d(\pi_2(\omega), \pi_1(\omega))} \tag{4.15}$$

$$\overline{d(\pi_1(\omega), \pi_2(\tilde{\omega}))} = \overline{d(\pi_2(\omega), \pi_1(\tilde{\omega}))} \tag{4.16}$$

and only one of each of those values needs to be reported in the following Tables 4.2, 4.3, and 4.4.

Table 4.2 contains the distances of the behaviour observed from the two agents trained with ARS, referenced as ARS 0 and ARS 1 in the figures before. Comparing these distance measurements, different observations can be made: If all distances are somewhat similar, no meaningful information can be extracted from the measurement. That is the case using 100 snapshots ("Snap 100") and using the last position ("Last Pos.") for a behavioural distance. If both policies are closer to themselves than they are compared to each other, the distance measure can recognise them as different. That seems to be the case using optimal transport of the occupancy measure ("OT") or dynamic time warping ("DTW") as a distance measure. But that is only visible in the mean with a large standard deviation over the 10 episodes.

Table 4.2: Compare π_1 and π_2 both trained with ARS under the same sample function ω or different sample functions ω or $\tilde{\omega}$. The values are the average $\pm$ standard deviation of 10 distances that are calculated considering one episode each as behaviour.

Metric	$\overline{d(\pi_1(\omega), \pi_1(\tilde{\omega}))}$	$\overline{d(\pi_2(\omega), \pi_2(\tilde{\omega}))}$	$\overline{d(\pi_1(\omega), \pi_2(\omega))}$	$\overline{d(\pi_1(\omega), \pi_2(\tilde{\omega}))}$
Snap. 100	24.5 ± 5.9	26.0 ± 7.3	26.9 ± 5.0	29.1 ± 5.1
Last Pos.	2.5 ± 1.6	2.6 ± 1.5	2.9 ± 1.3	2.5 ± 1.3
Sgnt. 3	693 ± 350	707 ± 360	461 ± 192	684 ± 273
OT	854 ± 1088	923 ± 1100	1108 ± 661	1114 ± 610
DTW	1148 ± 580	1198 ± 584	1452 ± 334	1541 ± 359

Table 4.3: Compare π_1 and π_2 both trained with ARS under the same sample function ω or different sample functions ω or $\tilde{\omega}$. The values are the average $\pm$ standard deviation of 10 distances that are calculated considering an aggregation of 10 episodes each as behaviour.

Metric	$\overline{d(\pi_1(\omega), \pi_1(\tilde{\omega}))}$	$\overline{d(\pi_2(\omega), \pi_2(\tilde{\omega}))}$	$\overline{d(\pi_1(\omega), \pi_2(\omega))}$	$\overline{d(\pi_1(\omega), \pi_2(\tilde{\omega}))}$
Snap. 100	7.2 ± 1.6	8.4 ± 2.0	11.1 ± 1.0	11.7 ± 1.1
Last Pos.	0.8 ± 0.5	0.8 ± 0.4	1.3 ± 0.3	1.2 ± 0.2
Sgnt. 3	167 ± 92	151 ± 72	153 ± 44	178 ± 64
OT	1969 ± 2117	2456 ± 2275	6490 ± 1243	6820 ± 1023
DTW	397 ± 147	435 ± 146	766 ± 78	798 ± 61

This indicates that random influences dominate the tendency towards different behaviour. The distance measure based on a signature transform with depth 3 ("Sgnt. 3") has all values comparable, except the distance of different policies with the same random initialisation. That indicates the opposite, that the random initialisation and not the policy makes the main difference in how the behaviour looks. If that were true, it should be reflected in all metrics, not just that one. So, this indicates a property of the metric rather than a property of the data.

Repeating the same analysis in Table 4.3 but with an aggregation of 10 episodes for each data point confirms this suspicion: Now all distance measures, except using signature transform, pick up on the differences between the policies. Also, the values are now often separated by more than one standard deviation, which suggests that the separation is robust against random fluctuations.

A comparison between Tables 4.2 and 4.3 also suggests that comparing distances of several episodes and averaging the measured distances is an inferior approach to aggregating episodes first and then comparing distances. The mean values in Table 4.2 are equivalent to the first approach, averaging 45 measured distances for $\overline{d(\pi_i(\omega), \pi_i(\tilde{\omega}))}$ with $i = 1, 2$ and $\overline{d(\pi_1(\omega), \pi_2(\tilde{\omega}))}$. They are barely different, which means that with this point of view, π_1 and π_2 cannot be distinguished by their behaviour. In Table 4.3, the individual data points that contribute to the averages reported correspond to the second approach for aggregating first and then measuring distances. The reported mean and standard deviations indicate that, at least for optimal transport and dynamic time warping, it is very unlikely to observe similar distances.

Comparing that to Table 4.4, which contains the distances of the behaviour of the two agents trained with PPO referenced in the figures before as PPO 0 and PPO 1. Here, all

Table 4.4: Compare π_1 and π_2 both trained with PPO under the same sample function ω or different sample functions ω or $\tilde{\omega}$. The values are the average $\pm$ standard deviation of 10 distances that are calculated considering an aggregation of 10 episodes each as behaviour.

Metric	$\overline{d(\pi_1(\omega), \pi_1(\tilde{\omega}))}$	$\overline{d(\pi_2(\omega), \pi_2(\tilde{\omega}))}$	$\overline{d(\pi_1(\omega), \pi_2(\omega))}$	$\overline{d(\pi_1(\omega), \pi_2(\tilde{\omega}))}$
Snap. 100	8.1 ± 2.5	7.7 ± 2.5	5.9 ± 0.3	8.1 ± 3.2
Last Pos.	0.9 ± 0.6	0.9 ± 0.5	0.6 ± 0.4	1.1 ± 0.7
Sgnt. 3	160 ± 80	163 ± 74	102 ± 54	186 ± 90
OT	3805 ± 3185	3528 ± 2886	629 ± 839	4900 ± 3938
DTW	473 ± 216	457 ± 207	251 ± 55	482 ± 263

metrics agree that the distance between one policy and the other under the same random seed for initialisation of episodes is closer than the distance of either policy to itself, thus implying clearly that both policies are close in behaviour. This shows that the choice of the initial random seed for initialisation of episodes is dominant over the choice of policy.

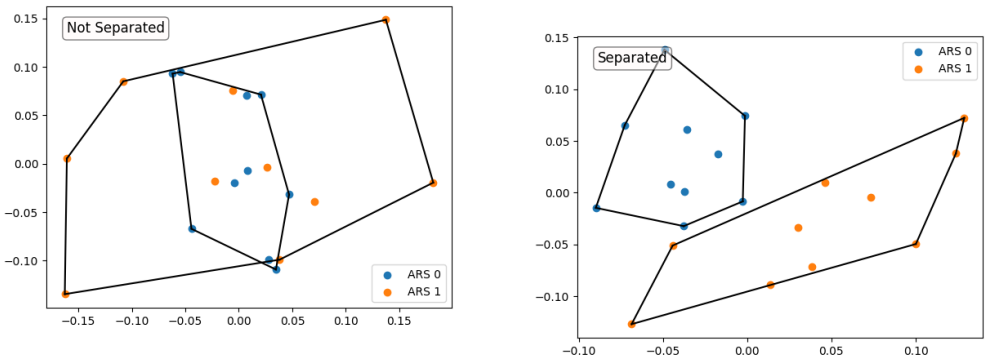
A practical conjecture can be inferred: Assume the "true" distance between the policies $d(\pi_1(\omega), \pi_2(\omega))$ using the same exact initialisation is not available, maybe because it is data that is collected from a real-world experiment. Then the conclusion that the policies are very close in behaviour can still be inferred from noting that the distance between the two policies is very close to the distance between either policy to itself, so if $\overline{d(\pi_1(\omega), \pi_1(\tilde{\omega}))} \approx \overline{d(\pi_1(\omega), \pi_2(\tilde{\omega}))}$.

Several points can be made from this exemplary exposition: Not every distance measure can pick up on differences between given behaviours. If they can, the differences can be visualised, and some understanding can be gained from that. It is also clear that the random seed for the initialisation of episodes significantly influences the behaviour. The observed patterns are only visible when several episodes are examined. For example, if only two points from Figure 4.4 were shown at random, it would be impossible to tell whether they came from the same policy or from different ones.

4.3.2 Separation of Policies

The idea that a behavioural distance can separate two policies can be formalised for a meta-metric that indicates the information value of a metric on policies. In this pairwise test, each policy is represented by 10 samples of its behaviour. The 20 samples, 10 for each policy, are then projected in a 2-dimensional space, just as they are for a visualisation. Such a configuration is called separated if the convex hulls of the points that belong to each policy are disjoint, see Figure 4.11.

This simple test allows an automated evaluation of different techniques to compare policies. Generally, a good metric should be able to separate as many policies as possible. From a given set of policies, it may not be possible or feasible to separate all policies. As the previous example on the two agents trained with PPO demonstrates, the random seed for



(a) Using 100 snapshots and the Euclidean distances

(b) Using optimal transport

Figure 4.11: Visualisation of the idea of separation with episodes collected from two agents trained with ARS. Distances are calculated based on one episode.

Table 4.5: The percentage of pairwise comparisons where the given metric separates two policies, depending on the behavioural distance and on the number of episodes that sample the behaviour of a policy.

Aggregation	1	5	10
Snap. 100	48.5	51.5	57.6
Last Pos.	7.6	31.8	36.4
Sgnt. 3	0	7.6	21.2
OT	71.2	75.8	87.9
DTW	69.7	75.8	77.3

the initialisation of episodes may have a stronger influence on the outcome than the choice of policy. These policies are then so close to each other that a separation would be a random artefact, not an actual representation of the data.

For this test, the same twelve policies for CartPole as in Section 4.2 are evaluated. Table 4.5 presents the results for different behavioural distances and different amounts of episodes used to characterise the behaviour.

In total, 66 pairwise comparisons are made. The most separations achieved are 58 out of 66 with optimal transport and 10 aggregations. It seems plausible that this representation retains the most information about the relationship of the policies' behaviour, also given the previous observations. In general, the percentage of separation increases with the number of aggregated episodes. Aggregating episodes smooths out the random fluctuations that occur in any episode, and the individual features of each policy become more pronounced. That seems to be beneficial for discriminating policies even when using behavioural descriptors that conceptually do not lend to averaging, such as selecting the final state.

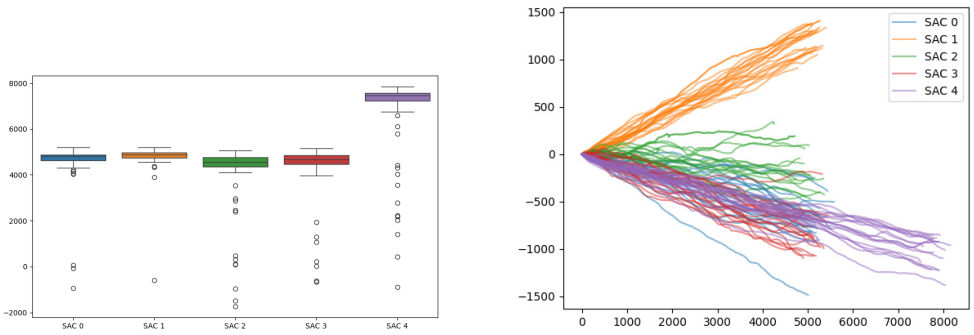
4.4 Applications

Having the principles of behavioural distance and visualisation of behavioural distance established on the example of some policies for the CartPole environment, the next step is to examine whether these principles hold in more complex environments. Then, a theoretical remark follows on the validity of the behavioural distance as an actual, mathematical distance. With the notion of robustness, an independent characterisation of policies is introduced that can be aligned with the behavioural distances to further validate the visualisation technique and to show that the behavioural distances actually manifest in tangible differences in the policies. An application for visualising policies is given in an analysis of the change of behaviour during a reinforcement learning algorithm, which solidifies the approach as a technique for explainable reinforcement learning.

4.4.1 Discriminating Ants

The CartPole environment implements a simple and low-dimensional task. A typical reinforcement learning task inspired by robotics further validates the ideas presented. The Ant environment allows further analysis of the behavioural metrics. Can this data-driven approach recognise an ant by its gait?

The environment is a partially known Markov decision process in 87 continuous state dimensions and 8 continuous action dimensions. The ant incurs a reward by walking in a given direction. Most of the state dimensions could be ignored: 60 state dimensions contain



(a) Performance boxplot **(b)** 10 physical paths

Figure 4.12: Naive characterisations of the behavioural data of the Ant policies. Left rewards over 100 episodes, right the physical path taken during episodes as seen from above (x- and y-coordinates).

information about the contact forces that the ant is experiencing. This information is mostly irrelevant to solving the problem of walking. This makes this a challenging problem for deep learning algorithms, but also for the challenge to distinguish behaviour based on this data. Since this problem is more complex, the baseline data is created from five different training runs that all use SAC [HZA+18], using the baseline implementation from Brax [FFR+21].

Here, it is more obvious to find naive characterisations of behaviour: The reward is not capped at a certain point, but rather, fast walkers will incur more reward. Also, the path taken can be visualised more easily, see Figure 4.12(b). It depicts the path of the simulated robot as seen from above. The reward signal is delivered when going right, in the positive direction of the x-axis.

Looking at Figure 4.12(a), the reward signal alone distinguishes one agent (SAC 4) from the others. This is reflected in Figure 4.12(b) where this agent's observed paths extend further. Another agent can also be clearly identified because it exhibits an upward drift up rather than a downward one.

The test of separation gives an indication of the information value of the different techniques for a data-driven distinction of the agents in Table 4.6. The naive approaches do not properly separate policies without aggregation, while the more sophisticated techniques mostly do. The signature transform does not separate SAC 0 and SAC 2, while dynamic time warping does not separate SAC 2 and SAC 4. After aggregation, almost every constellation is separated, except for the last position when SAC 2 is compared with SAC 0 or with SAC 3.

Table 4.6: Percentage of pairwise comparisons where the given metric separates two policies.

Aggregation	1	10
100 snapshots	20	100
Final state	10	80
Signature, depth 3	90	100
Occupancy measure	100	100
Dynamic time warping	90	100

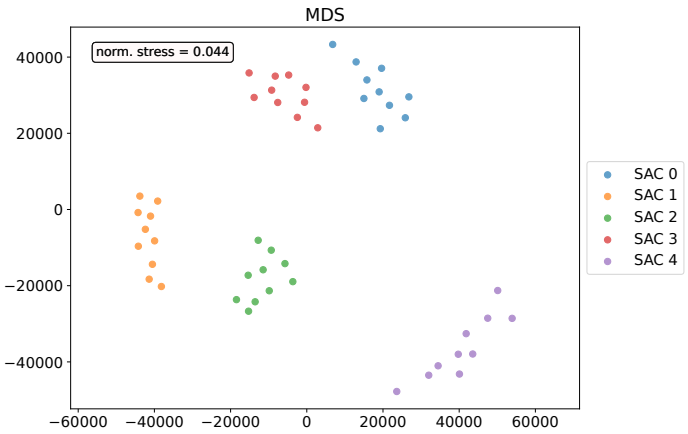


Figure 4.13: Visualisation of all 5 policies using signature transform, 10 aggregations, and MDS. Each different colour represents a different policy.

Visualising all five policies in one plot (Figure 4.13) with aggregation shows a clearer picture of the relationships, also reflected in the naive visualisations in Figure 4.12. All policies are separable. SAC 0 and SAC 3, which were indistinguishable in the naive representation, are the closest but clearly separate. SAC 4 is the most distant from the policies. Whatever makes this policy walk faster than the other four policies, which all walk with roughly the same speed, seems to be picked up by the distance measure.

4.4.2 A Proper Metric

Let $\mathcal{M}$ be a finite Markov decision process, $\mathcal{T}$ be its space of episodes and π be a policy for it. Following Theorem 1, a random element exists that represents the process of rolling out episodes with π, which abusing notation is also called π here. Different from the previous analysis in this chapter, both the policy and the Markov decision process may also be stochastic which is incorporated in that random element.

A more rigorous definition of a metric can be achieved by defining a sample function ω with $\pi(\omega) = \tau$. In that case, given a metric on the space of trajectories, $d : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ induces a metric in the space of policies without differences that are not realised in behaviour, i.e., behaviour in states that are never visited:

$$d(\pi^{(1)}, \pi^{(2)}) = \mathbb{E}\left[d(\pi^{(1)}(\omega), \pi^{(2)}(\omega)) \mid \omega \in \Omega\right] \tag{4.17}$$

As noted before in Subsection 4.3.1, for all instances of behavioural distances introduced in this chapter, this quantity is 0 for $\pi^{(1)} \equiv \pi^{(2)}$. This is not by coincidence. When fixing the random seed, or rather the sample function, this definition makes a proper metric:

Theorem 9. *Let $\mathcal{M}$ be a Markov decision process with finite state space, action space, and time horizon. Let $d_{\mathcal{T}} : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ be a metric on $\mathcal{T}$, the space of episodes in $\mathcal{M}$. Then $d_{\Pi} : \Pi \times \Pi \to \mathbb{R}$ defined by*

$$d_{\Pi}(\pi, \tilde{\pi}) = \mathbb{E}\left[d_{\mathcal{T}}(\pi(\omega), \tilde{\pi}(\omega)) \mid \omega \in \Omega\right] \tag{4.18}$$

is a metric on Π the space of policies.

Proof. Show properties of a metric are satisfied:

1. Since $d_{\mathcal{T}}$ is a metric, $d_{\mathcal{T}}(\pi(\omega), \pi(\omega)) \equiv 0$ for all $\omega \in \Omega$. It follows that $d_{\Pi}(\pi, \pi) = 0$.
2. Let $\pi \neq \tilde{\pi}$ be behaviourally different. Then there exists at least one sample function ω_0 such that $\pi(\omega_0) \neq \tilde{\pi}(\omega_0)$. Since $d_{\mathcal{T}}$ is a metric, $d_{\mathcal{T}} \geq 0$ holds and $d_{\mathcal{T}}(\pi(\omega_0), \tilde{\pi}(\omega_0)) > 0$ and so $d_{\Pi}(\pi, \tilde{\pi}) > 0$.
3. $d_{\mathcal{T}}$ is a metric, so $d_{\mathcal{T}}(\pi(\omega), \tilde{\pi}(\omega)) = d_{\mathcal{T}}(\tilde{\pi}(\omega), \pi(\omega))$ for every $\omega \in \Omega$. It follows that $d_{\Pi}(\pi, \tilde{\pi}) = d_{\Pi}(\tilde{\pi}, \pi)$.
4. Let $\pi_{x,y,z}$ be three policies. Then

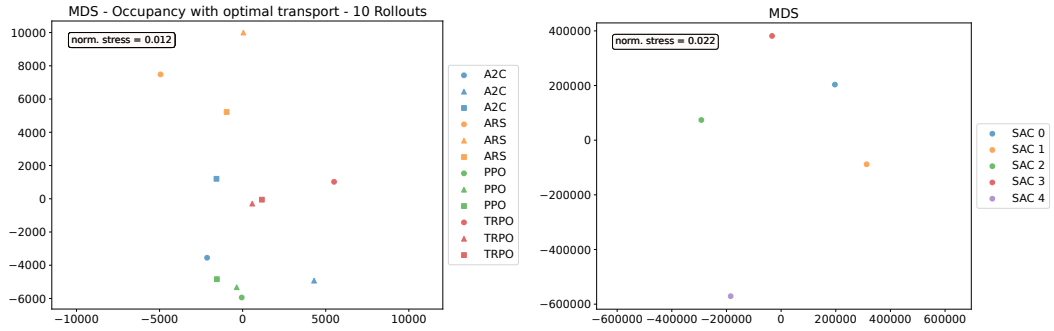
$$\begin{aligned} d_{\Pi}(\pi, \tilde{\pi}) &= \mathbb{E}\left[d_{\mathcal{T}}(\pi_x(\omega), \pi_z(\omega)) \mid \omega \in \Omega\right] \\ &\leq \mathbb{E}\left[d_{\mathcal{T}}(\pi_x(\omega), \pi_y(\omega)) + d_{\mathcal{T}}(\pi_y(\omega), \pi_z(\omega)) \mid \omega \in \Omega\right] \\ &= d_{\Pi}(\pi_x, \pi_y) + d_{\Pi}(\pi_y, \pi_z) \end{aligned}$$

□

With this definition, the agents from before can again be measured in distance and visualised. The proper metric, of course, does not allow the test of checking the similarity of a policy to itself. The distance measure does not have any variance in self-similarity anymore. Using MDS again to plot the calculated distances in Figure 4.14, the relationship between the different policies coincides with the previous analysis.

For the CartPole environment, two policies trained with TRPO are very similar in one cluster, and all the policies trained with PPO and one trained with A2C are very similar in another cluster. The other policies behave differently from each other and the clusters. Also, for all but A2C, policies trained with the same learning algorithms are close to each other. The plot of the five policies for the Ant environment clearly separates the policies. However, SAC 0 and SAC 3, the two policies that are indistinguishable from their paths, are closest, and SAC 4, the policy that accumulates higher rewards, is the most distant from the cohort.

It is evident now that this combination of transformation and aggregation through the occupancy measure can pick up differences in the behaviour of policies, such that it is possible to classify behaviour to the agent that produced it and to some extent cluster different



(a) CartPole policies **(b)** Ant policies

Figure 4.14: Plot of the previously analysed policies based on a proper metric using 10 aggregations and optimal transport. Left for the agents on CartPole, right on Ant.

agents in a way that reflects the algorithm that created the agent. This definitely shows that different types of behaviours are created depending on the algorithm that created it, and that certain algorithmic learning setups will gravitate to certain types of solutions. It also shows that agents have a recognition value that makes it possible to recognise their specific behaviour from a dataset.

While this technique, comparing only episodes that use the same random seed, has improved the clarity of the results, it is not the most practical. In a real-world scenario and even in many simulation environments it is not possible to fix and repeat random influences. The increased clarity does encourage trust in the validity of the approach and may be of further theoretical interest, but for a general practical application, it may be too restrictive.

4.4.3 Robustness

It is clearly possible to pick up differences in behaviour from policies. It is still up for debate if these differences are in any way meaningful. The *robustness score* [FPG24] offers a practical approach to extract additional information about policies regarding their performance.

The *robustness* of a policy refers to its ability to adapt to unexpected environmental changes: Can a policy trained in one setup of the environment still perform when the setup of the environment changes? In the CartPole environment, several parameters are fixed at certain values. For this investigation, gravity and pole length are chosen to be varied.

Definition 24 (Modified CartPole)**.** *Let $\mathcal{M}^{cart}_{g,l}$ be the Markov decision process that is created by changing the gravity in the CartPole environment to the value g and pole length to the value l.*

Call $\eta(\pi|\mathcal{M}^{cart}_{g,l})$ the performance (according to Definition 15) of the policy π in the environment $\mathcal{M}^{cart}_{g,l}$.

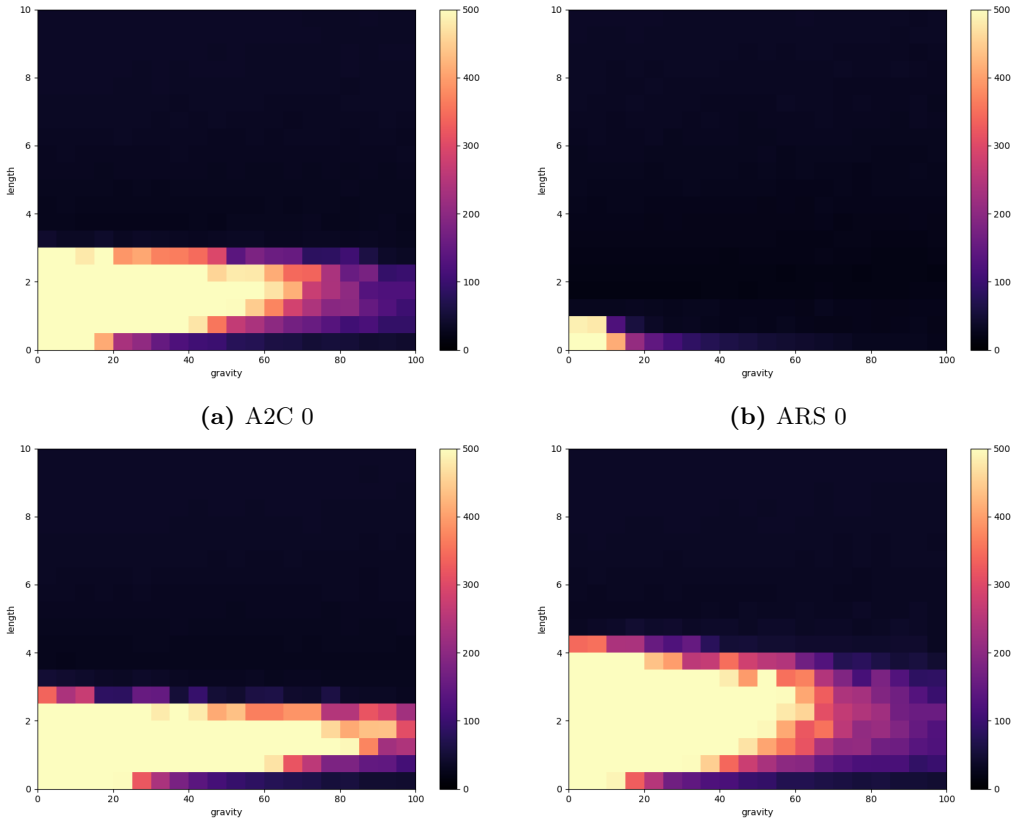
From a 20×20 grid $(g_i, l_j)_{i,j=1,\dots,20}$ with values $g_i = 5i - 2.5$ and $l_j = 0.5j - 0.25$, in total 400 new environments are created and for a given policy π its performance $\eta(\pi|\mathcal{M}^{\text{cart}}_{g,l})$ is evaluated over all of them. This can be visualised as a *robustness profile* of a policy π: A heatmap, that colours a square around the point (g, l) with a colour corresponding to the polices performance $\eta(\pi|\mathcal{M}^{\text{cart}}_{g,l})$ gives a visual understanding how the policy copes with changes of the given variables in the environment, for example in Figure 4.15.

This profile already shows that all four policies are actually different in a meaningful way, in a way that may influence their usefulness in a real setting, where the environment may change over time or with different applications. Some policies are simply better than others concerning this robustness, like the A2C policy that completely dominates the ARS policy. But also between the PPO and TRPO policies, a trade-off can be observed, where the PPO policy is still strong in high-gravity situations, but the TRPO policy can handle a longer pole. To reduce this rather complex profile to a single number, a *robustness score* is defined:

Definition 25 (Robustness score)**.** *For a policy $\pi : \mathbb{R}^4 \to \{\pm 1\}$ that functions with the CartPole environment, call the* robustness score *$rs(\pi)$ a sum of the policy's performance over 400 modified CartPole environments*

$$rs(\pi) = \sum_{i,j=1}^{20} \eta\left(\pi|\mathcal{M}^{cart}_{g_i,l_j}\right), \tag{4.19}$$

with modified gravity values $g_i = 5i - 2.5$ and modified pole length values $l_j = 0.5j - 0.25$.



(a) A2C 0 **(b)** ARS 0

(c) PPO 0 **(d)** TRPO 0

Figure 4.15: Robustness profiles based on changes in pole length and gravity of the agents trained on the default CartPole that were previously visualised as A2C, ARS, PPO, and TRPO 0.

This robustness score is applied to the 12 CartPole policies that were analysed before in Figure 4.16. It is overlaid on the final visualisation shown in Figure 4.14(a). It shows that there is a clear relationship between the positioning of a policy in the visualisation and the calculated robustness score: The three ARS policies that are far away all have a low score around 15000. The one TRPO policy that is away from the other has, with 62000 points, a much higher score than the rest of the policies that all fall somewhere between 37000 and just below 50000. The visualisation already has these three regions of robustness score separated. The visualisation was based solely on behaviour in the original, unmodified environment, without any knowledge of how the agents would behave in the modified environments. While it shows pairs of policies very close to each other that differ by a robustness score of up to 10000, so by 20-25% of their value, the three policies with the lowest robustness score are well separated from the rest, and the one outlier with a score over 60000 is also separated from the rest. That indicates that the difference between policies calculated in this way reflects a real, meaningful difference in behaviour that captures relevant properties of the policies.

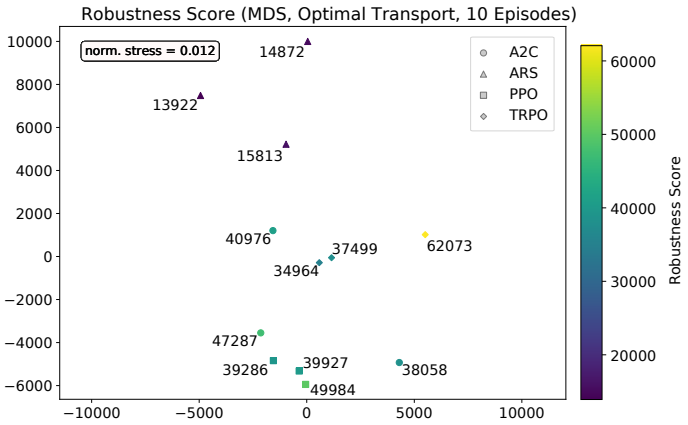


Figure 4.16: Visualisation of the previously analysed policies for CartPole using 10 episodes with matching random seed, and optimal transport mirroring Figure 4.14(a). Colour-coded and annotated to show their robustness score.

4.4.4 Observing the Learning Process

To demonstrate applications of visualising policies, the learning process of a run of the PPO algorithm for the CartPole environment is observed by persisting the model parameters every 2048 steps in the environment, which amounts to one policy saved at every four gradient steps. This is done 50 times, resulting in a list of 50 different agents. Typically, such a learning process is observed with the average reward accumulated and the loss with changes to the parameters, see Figure 4.17. According to this figure, good solutions are found with policies 17-25. Then, some catastrophic change happens, and the algorithm converges around policy 37 when the reward is at maximum, and the learning loss is close to zero.

In line with the ideas of this chapter, additionally, each agent is rolled out 10 times, and then the established distance measure, using optimal transport and the same seed, is used to calculate pairwise distances. To illuminate the proper solution space, only those agents that achieve full performance in all 10 episodes are plotted in Figure 4.18. Further, for each agent, calculate the robustness score following Defintion 25. This colour codes the individual agents. The points are annotated with a number that shows when during the learning process the agent was recorded.

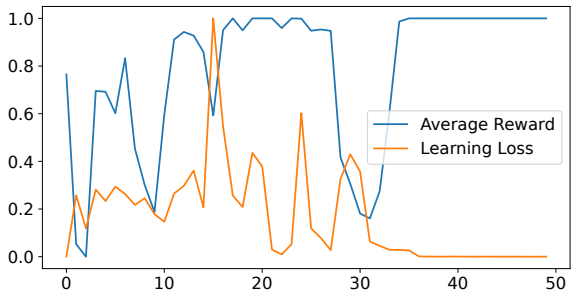


Figure 4.17: PPO's learning process for CartPole observed with the standard metrics: average accumulated reward as the measure of performance and learning loss as the measure of change. Min-Max-Scaled.

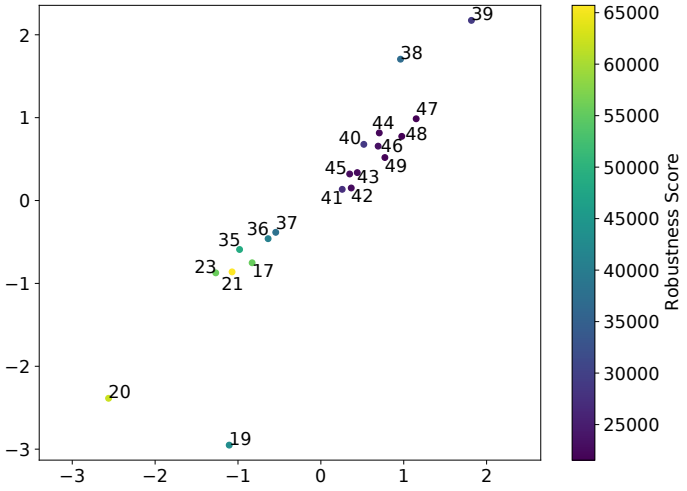


Figure 4.18: A visualisation of all the perfectly performing agents encountered during learning CartPole with PPO, colour-coded by robustness. The visualisation is done using MDS on the average of pairwise optimal transport distances of 10 episodes created with the same random seeds.

The resulting visualisation in Figure 4.18 displays two clusters: The first cluster consists of the models 17, 21, 23, 35, 36, 37. Watching the episodes with human eyes reveals that these policies all work by wiggling the cart to keep the pole upright, but with a slight and steady drift towards the left. The outliers 19 and 20 have a similar behaviour, but at some point they reverse that drift towards the right and get faster. The other cluster consists of models 40-49, all of them wiggle very tightly to hold the pole perfectly straight and drift very little from the middle of the screen. The outliers 38 and 39 move left first, tilting the pole to the right and then slowly moving right and wiggling to keep the pole straight. They are behaviourally the opposite of the first cluster, and it seems that combining the experience from the first cluster with that experience lays the foundation for the very tight behaviour seen in the second cluster.

The robustness analysis mirrors the clustering to some degree: The first cluster has robustness scores between 37860 and 65712, with a somewhat clear gradient, the highest being model 21, the farthest away from the other cluster, 17 and 23 have almost the same score with 55429 and 55682, and 35, 36, 37 have decreasing scores with 48553, 40581, 37860, as they get closer to the second cluster. This second cluster with models 40-49 only has robustness scores between 21549 and 28921. The outliers close to that cluster, models 38 and 39, also have a score under 30000, more closely aligned with the second cluster. The outliers, models 19 and 20, have higher scores, 42983 and 62160, respectively, more closely aligned with the first cluster.

Both the visual inspection and the robustness analysis of these models and clusters validate that the visualisation technique that was derived previously in this chapter meaningfully separates and clusters behaviour. But what can be learned from this visualisation about the learning process?

- There are several optimal policies for CartPole and several of them are found during

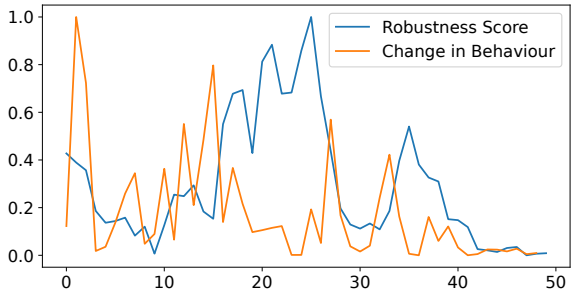


Figure 4.19: The same learning process as in Figure 4.17 observed with by logging the change in behaviour and the robustness scores during learning: The robustness score as a measure of secondary performance and the change of behaviour measured as the distance of this agent to the previous one. Min-Max-Scaled.

one run of a learning algorithm, since Figure 4.18 contains only perfectly performing agents, which differ clearly in their behaviour.

- The learning process continues with significant changes even if an optimal policy is found, as can be seen by the progression of the solutions 36-40 in Figure 4.18.
- The visual inspection earlier in this Subsection suggests that there is a specific type of optimal policy that the learning algorithm converges to. This preferred type of optimal behaviour is characterised by maintaining states that are as different as possible from terminal states.
- The robustness scores in Figure 4.18 show that convergence to this preferred optimal policy constitutes overfitting. The capability to generalise to changed environments is sacrificed for a strategy that is strongly aligned to the original environment.

When considering the change in behaviour and the robustness scores during learning, different aspects of the learning process are unveiled. In Figure 4.19, this is depicted in a line plot reminiscent of the earlier plot in Figure 4.17. Many significant changes in the beginning (policies 1-17) lead to a somewhat stable region (17-28) with high robustness. Then an important change happens that leads to a low robustness area (corresponding to the performance dip) around policy 30. This is followed by another big change, which is followed by the performance and robustness boost around policy 35, but then smaller changes keep occurring until policy 47. This is significantly longer than the loss or the performance in Figure 4.17 implies, and the changes in behaviour are accompanied by a measurable impact on robustness. It becomes clear that the learning did not stop when the loss became very small, but rather that was when the most significant overfitting began. This is possible because the relationship between phenotype and genotype is chaotic, so small changes in the parameter space may still have large consequences in the behaviour space, refer also to Chapter 3.

4.5 Why Visualise?

This chapter demonstrates how, through episodic and occupancy understanding of a policy, a distance based on behaviour can be constructed that differentiates them in a meaningful way without injecting specific expectations into the problem. With that, new questions

can be asked: Do certain algorithms always find the same, or rather always similar types of solutions? Are certain algorithms more similar in the types of solutions they find? And from the visualisation in this chapter, at least some indications of those questions can be derived. It seems reasonable to assume that some algorithms converge more clearly towards certain solutions, that there is some form of self-similarity in the outcomes of algorithms, and that the behaviour of a policy trained from one algorithm is expected to be closer to the behaviour of a policy trained with the same algorithm than one that was trained with another algorithm.

With this kind of questioning, a more detailed dissemination of reinforcement learning algorithms and their components is possible. The typical metrics for reinforcement learning are performance and cost until convergence is reached, measured in quantities such as compute time or steps in the environment. But a difference in behaviour may be useful to better understand the character of the underlying problem and how the learning algorithm relates to it.

For example, the analysis of the learning process in Figure 4.18 and the visual inspection in Subsection 4.4.4 show that one instance of PPO applied to the CartPole environment converges to a behaviour that tends to maintain certain preferred states far away from harmful states even after finding a different optimal strategy. The behavioural analysis of multiple agents trained with PPO, only using different random seeds for random number generation during the training process, in Figure 4.14(a) shows that PPO tends to develop similar policies when restarting. Combining both observations suggests an underlying property of the PPO algorithm: That in a survival task like CartPole, PPO prioritises maintaining certain states which may lead to overfitting.

Understanding similarity and difference is a core aspect of understanding a class of items. It is an aspect that is both central to the theory and practice of reinforcement learning and, at the same time, a gravely overlooked concept in the pursuit of a better understanding of the practical problems and outcomes in reinforcement learning. This chapter has shown new pathways towards such an understanding.

CHAPTER 5

Evolutionary Computation with Diversity

The behavioural understanding of policies is an integral part of the theory of reinforcement learning and in the derivation of its algorithms. There it comes into play in a very abstract manner. In evolutionary computation, another angle on differentiating policies is found. With a fundamentally different approach on optimisation, it offers natural extensions that find iterative improvements not just based on the optimisation target but also on diversity. Defining and especially quantifying diversity is rooted in finding differences between solutions. In the realm of evolutionary computation, another, very practical viewpoint on the difference between policies exists. It has already inspired a baseline for visualisation in Subsection 4.2.2 with the behavioural descriptors "last state" and "K snapshots". But with new, unbiased methods to differentiate behaviour, it is tempting to analyse whether these methods that differentiate behaviour can also be used to foster diversity in an algorithm that needs a mechanism to distinguish policies.

Diversity can mean both genotypical and phenotypical differences. In the context of reinforcement learning, phenotypical differences seem more promising, as argued before. This chapter gives an overview of different methods in evolutionary computation that utilise some form of behavioural differences in finding solutions to reinforcement learning problems. Section 5.1 introduces evolutionary computation, showcases two methods, NEAT and novelty search, and the concept of quality-diversity. Section 5.2 is dedicated to the MAP-Elites algorithm and its different components. These algorithms are of interest because they include ways to define and utilise behavioural distances. How that behavioural difference is defined is often as simply as possible and problem-specific, with a focus on defining novel algorithms that solve specific problems. But they also offer a chance to test the validity of more abstract forms of behavioural differences. Section 5.3 reflects on the value of diversity and novelty and what to expect when shifting the approach to behavioural difference to a general, problem-agnostic interpretation.

5.1 Evolutionary Computation

Evolutionary computation is an umbrella term for the fields of *genetic algorithms*, *genetic programming*, *evolutionary strategies*, and *evolutionary programming* [BKS+23]. They are a group of optimisation techniques that rely on metaheuristics inspired by biological evolution. They rely on the creation and maintenance of a *population* of candidate solutions, which, after initialisation, are improved in an evolutionary loop. This loop consists of a mechanism to create new candidate solutions (offspring or children) based on the existing solutions (parents) through *variation operators* such as *mutation* or *recombination*, which are also categorised as asexual and sexual reproduction, respectively. And, some form of *selection* mechanism (competition) that decides which solutions form the next iteration of the population. Each such loop is a *generation*.

Algorithm 3 Generic evolutionary computation algorithm

Initialise population X, evaluate $f(X)$.
for each generation **do**
  X' ← Offspring of X
  $f(X')$ ← Evaluation of X'
  X ← Selection from (X, X') based on $(f(X), f(X'))$

Evolutionary algorithms are generally gradient-free optimisation methods that can tackle the same problems and produce agents of the same structure as reinforcement learning algorithms. Details differ depending on the concrete heuristic, but generally an evolutionary algorithm is a learning algorithm as described in Algorithm 3. After initialisation, it consists of a loop of three key elements: The members of the population reproduce to create a larger pool of agents, a fitness function evaluates a population of agents, and a new, hopefully better-performing population of agents is selected. Higher fitness is associated with a higher chance of survival and reproduction. Applied to reinforcement learning problems, the fitness function is often the performance of a policy, but it can also include other factors.

Evolution strategies (ES) are an important branch of evolutionary algorithms. Sometimes they are classified with the addition (μ, λ) or $(\mu + \lambda)$. Here $\mu \in \mathbb{N}$ is the number of parents and $\lambda \in \mathbb{N}$ is the number of offspring in each generation. In (μ, λ), only the offspring compete in the next generation, while $(\mu + \lambda)$ implies that parents and children compete for the next generation. The creation of offspring further differentiates algorithms. The earliest implementation of this idea would use random mutation, that is, the addition of a sample of the normal distribution $\mathcal{N}(0, 1)$ to each value of a copy of the genotype of a randomly selected parent's genotype, to create offspring [Rec73]. *Neuroevolution* is a subset of ES that is concerned with the evolution of the architecture and the parametrisation of a neural network.

The addition of covariance matrix adaptation (CMA) that leads to the algorithm *CMA-ES* [HO01] proved very successful. Random mutations to create offspring are drawn from a multivariate normal distribution $\mathcal{N}(m, C)$. The mean m and the covariance matrix are initialised as the 0-vector and identity matrix. Over generations, m and C are updated to maximise the likelihood of drawing viable offspring based on past experience. This way, the sample updates of the solution vector adapt to the solution space of the optimisation problem and are shifted, scaled, and interconnected appropriately.

Genetic Algorithms, on the other hand, are more concerned with more complicated encodings and crossover operators than evolution strategies. While an evolutionary strategy searches a parameter space in a fixed function architecture, a genetic algorithm would also build the architecture of the function. The addition of *programming* to either of the two ideas implies the creation of some instructions that then address the given tasks instead of a function that directly tackles them.

Two main distinctions in propagating to the next generation are asexual reproduction with new agents arising from one agent by mutating, that is, copying and randomly changing an agent, or by sexual reproduction that is randomly crossing the architecture and inner makings of two agents and then maybe mutating them to create a new agent. Further, evolutionary algorithms differ in how the population and reproduction are managed based

on performance. And most importantly, the nature and architecture of the agents are a crucial ingredient of any evolutionary algorithm, as well as several hyperparameters of the algorithm.

5.1.1 NEAT

The *NeuroEvolution of Augmenting Topologies* algorithm or *NEAT* [SM02] is an important milestone in evolutionary algorithms, as it combined important ideas in the community, beat existing benchmarks in a simple, elegant algorithm, and spurred invention with several derived learning algorithms.

The algorithm constructs neural networks that serve as agents to solve a control problem. It puts value in diversity in the solution space and incorporates the search for an optimal topology of the agent into the optimisation process instead of leaving it to the user. At the same time, it manages to avoid problems that plagued evolutionary algorithms at the time: It avoids the "competing conventions problem" or the "permutations problem", a particularly significant problem when sexually reproducing or crossing agents. This problem arises because the same strategy can be completely differently encoded in numerous ways in the same topology of a deep neural network. That means, while crossing the behaviour of two agents might produce an even better behaviour, crossing their inner encoding might give rise to a complete nonsensical behaviour in the resulting agent. To avoid this, NEAT uses "artificial synapses".

To encourage diversity and protect innovation, NEAT utilises "explicit fitness sharing" with speciation. Organisms mate and reproduce within their species, and organisms in a species share their fitness payoff, the opportunity to reproduce. Similarly, what defines speciation is set in terms of encoding, not behavioural comparison.

5.1.2 Novelty Search

From the area of evolutionary algorithms, *novelty search* [LS08] presents an open-ended approach to optimisation that specifically aims at overcoming deception and local optima. It is originally built on the NEAT method [SM02]. It introduces the *novelty score* based on the behaviour of an agent, described by a behavioural descriptor (see Definition 18).

Given a behavioural descriptor and a list of previous solutions (an *archive*), the novelty score is defined with a k-nearest neighbour approach: The novelty score of a new solution is the average distance of its behaviour characteristic to the k nearest behavioural descriptors of the solutions in the archive. The archive is initialised with random solutions. Instead of using the performance of a solution for fitness, novelty search uses the novelty score. The evolutionary loop of NEAT creates new solutions based on solutions in the archive, and if these new solutions exceed a threshold, they are added to the archive. For an algorithmic description, see Algorithm 4. It is worth noting, that the NEAT component is not seen as essential to understand an algorithm as a type of novelty search [CMS+18].

Novelty search successfully solves a deceptive maze [SM02], a representative of the problem class of hard exploration problems (see also Subsection 2.3.4) that are difficult to solve for standard reinforcement learning techniques: In this maze, the reward function is defined as the negative of the distance to a goal. The closer the agent gets to the goal, the bigger the reward. However, there is a dead-end that is close to the goal and easy to reach. Going there is a local minimum that standard optimisation methods fall in and typically cannot escape. The actual goal can be reached, but it requires the agent to walk a route that initially offers

Algorithm 4 Novelty search algorithm

Initialise population X, evaluate behavioural descriptors $\phi(\tau_x)$ where $\tau_x \sim \pi_x$ for $x \in X$.
for each generation **do**
  $X' \leftarrow$ Offspring of X based on NEAT
  $\phi(X') \leftarrow$ Evaluation of behavioural descriptors of X'
  $f(X') \leftarrow$ Novelty of $\phi(X')$ based on $\phi(X)$
  $X \leftarrow X \cup \{x \in X' \mid f(x) > \text{ threshold}\}$

a smaller reward. In this application, the authors use the coordinates of the end step of the deterministic rollout of the policy as a characterisation of the behaviour of the agent. Taking the end step as a characterisation is an implicit injection of knowledge about the problem. While for mazes it is clearly very relevant what end step is reached, for an Atari game, it might be completely meaningless. In a second application, the authors achieve significantly better performance for a biped walker with Novelty Search than with NEAT. The behaviour of an agent is described by taking the evolution of the planar centre of mass of the walker as a vector. The distance between two behaviours is the Euclidean norm of their difference. This shows that focus on novelty may even improve performance in problem settings that are not adverse to gradient-based optimisation strategies.

The authors acknowledge the limitation imposed by handcrafting the diversity measure for the problem. It is unclear how more complex problems define diversity or what constitutes a useful diversity measure. Finally, they hint at an important next step for this endeavor, merging the diversity-seeking idea described in the paper to find promising strategies in the global solution space with classic optimisation methods to achieve locally optimised strategies. The family of algorithms that combine both is generally known as *quality-diversity* algorithms.

5.1.3 Quality-Diversity

Novelty alone can fail if the space that is being explored is too big in comparison to the space of viable solutions. Quality-diversity (QD) algorithms [PSS+15] reconcile the drive to explore new candidate solutions with the necessity to constrain the search to just a subset of possible solutions to increase sample efficiency, as they aim to find populations of candidate solutions that are diverse in behaviour but still high-performing. They tend to ignore the large part of the solution space that contains low-performing configurations.

Novelty search with local competition [LS11a] addresses this problem with a multi-objective optimisation procedure to optimise three objectives simultaneously: novelty score as before, but in this concrete instance of experiments considering genotypical diversity based on the evolved architecture of the small neural networks, and a local competition score, defined as the count of k-nearest neighbours in the novelty archive that the new solution outperforms. This represents a dynamic niche strategy. The performance of a new solution is not evaluated in relation to the global optimisation objective, but in relation to the performance of other, similar solutions. This strategy is therefore typically identified also as quality-diversity.

A later study that scales the idea of novelty search to large neural networks [CMS+18] through massive parallelisation simply replaces the objective with a weighted sum of the objective and novelty score. This mixed objective is then optimised with an evolutionary

strategy, a population-based random search, as the novelty score is not differentiable and a gradient-based method cannot be used.

5.2 MAP-Elites

The MAP-Elites (Multi-dimensional Archive of Phenotypic Elites) algorithm [MC15] tries to maintain the advantages of novelty search while mitigating the disadvantage of spending many resources on low-performing candidate solutions by constraining the evolutionary search to the *elite hypervolume*, the subspace of high-performing solutions spread along different types of behaviour [VM18]. For this purpose, behavioural *niches* are statically defined at the beginning of the optimisation process in a Cartesian grid: The image of the behavioural descriptor ϕ is bounded or clipped to a hypercube $\Phi \subset \mathbb{R}^m$, which is sliced into regular disjunct subsets Φ_i, where each subsection represents an evolutionary niche.

$$\phi : (S \times A)^\infty \to \Phi \tag{5.1}$$

$$\Phi = \dot{\bigcup}_{i=1,\dots,n} \Phi_i \tag{5.2}$$

The optimisation loop of MAP-Elites described in Algorithm 5 is that of a regular evolutionary algorithm. Its specific nature lies in the way the population is maintained. Any new solution is assigned a niche by binning the value of the solution's behavioural characteristic. The population, or the *archive*, is defined by holding at most one solution, the *elite*, in each niche. If a solution is assigned to an already occupied niche, then the new solution only replaces the existing solution if its performance is higher. This mechanism represents a form of local competition. MAP-Elites does not look for the one, best solution π^*, but for a population of elite solutions, the elite hypervolume, a collection of best performing individuals for each niche:

$$\Pi^*_\Phi = \left\{ \arg\max_\pi \left(\eta(\pi) | \phi(\pi) \in \Phi_i \right) \right\}_{i=1,\dots,n} \tag{5.3}$$

The arg max may not be uniquely defined for each niche, in which case a random choice is made. Depending on how the behaviour space is defined and partitioned, for some niches there may not exist any policy that fits. In that case, the niche may be left empty. Typically, the hypercube and with it the codomain of the behaviour characteristic is low-dimensional to manage the amount of niches, and most often a 2-dimensional descriptor space is chosen to enable a visual representation of the resulting population as a heat map, where colour represents performance.

MAP-Elites has proven to be a very resilient and widely applicable algorithm, also for other applications such as generating varied architectural designs [SH23]. Consequently, numerous variants and tweaks have been proposed. They include modifications to the archive, such as with centroidal Voronoi tessellation, more sophisticated ways to preprocess the behavioural descriptor for example with dimensionality reduction, and improvements to the creation of new candidate solutions for example by including gradient information.

5.2.1 Centroidal Voronoi Tessellation

An extension of MAP-Elites to higher-dimensional spaces is possible with the *Centroidal Voronoi Tessellation* [VCM18] approach, short CVT-MAP-Elites, or CVT-ME. Here, the

Algorithm 5 MAP-Elites algorithm

Initialise population X.
Evaluate behavioural descriptors $\phi(\tau_x)$ where $\tau_x \sim \pi_x$ for $x \in X$.
Create a partition $\dot{\bigcup}_{i=1,\dots,n} \Phi_i = \Phi$ of the image of ϕ.
for each generation **do**
  X' $\leftarrow$ Offspring of X using emitters
  $f(X'), \phi(X')$ $\leftarrow$ Evaluation of X'
  X $\leftarrow \{\arg\max_{x \in X, X', \phi(x) \in \Phi_i} f(x)\}_{i=1,\dots,n}$

niches are created by defining a set of *centroids*, points in the bounded behaviour space Φ that form centres of a Voronoi tessellation of the space. Centroids are created by uniformly sampling the space $n_{\text{cvt}}^{\text{sample}}$ times and, using K-means clustering, selecting a predefined number n_{cvt} of centre points $\{c_i\}_{i=1,\dots,n_{\text{cvt}}}$, see Figure 5.1 for a visualisation of the process. The space may be sampled uniformly, or behavioural descriptors recorded in a prior run may be used to sample more heavily the sections of the behavioural space that are more relevant. The space is then divided into cells

$$\Phi_i = \left\{ x \in \Phi : \|x - c_i\|_2 \leq \|x - c_j\|_2 \; \forall j \neq i \right\}. \tag{5.4}$$

The boundary regions of cells are not clearly assigned to one cell over the other. To achieve a proper disjoint union, assign the points on the boundary region between two cells to the cell with the lower index.

The authors note, that it may be more appropriate to use a fractional norm in high-dimensional spaces (see also [AHK01]), but they could not determine a measurable effect of switching from the Euclidean norm. So, the construction with the Euclidean norm and the K-means algorithm has become the default setting for this technique [CLB+24].

Each centroid represents a cell, which represents an evolutionary niche. The optimisation loop is the same as in MAP-Elites, and new solutions are assigned to the niche associated with the centroid closest to the solution's behaviour characteristic. This avoids the exponential scaling of niches when the dimension of the behavioural descriptor increases: A Cartesian grid with 10 subsections per dimension has 10^N cells, where N is the dimension of the domain. With a centroidal Voronoi tessellation, the number of cells or niches can be set to any amount, and it is possible to create more niches where more interesting behaviour is expected in a type of premeditated mesh refinement.

5.2.2 Dimensionality Reduction

While a CVT archive already enables higher-dimensional behaviour spaces, dimensionality reduction can be an important building block to reduce the complexity of the behavioural descriptor. With the full observed behaviour as a basis, dimensionality reduction can be an avenue for a generalised behavioural descriptor. The result can also be used as a behaviour descriptor, as demonstrated by AURORA [GC22b], where a concrete, but abstract low-dimensional representation of the behaviour is encoded in the latent space of an autoencoder that is evolved in parallel to the usual evolutionary iterations of MAP-Elites.

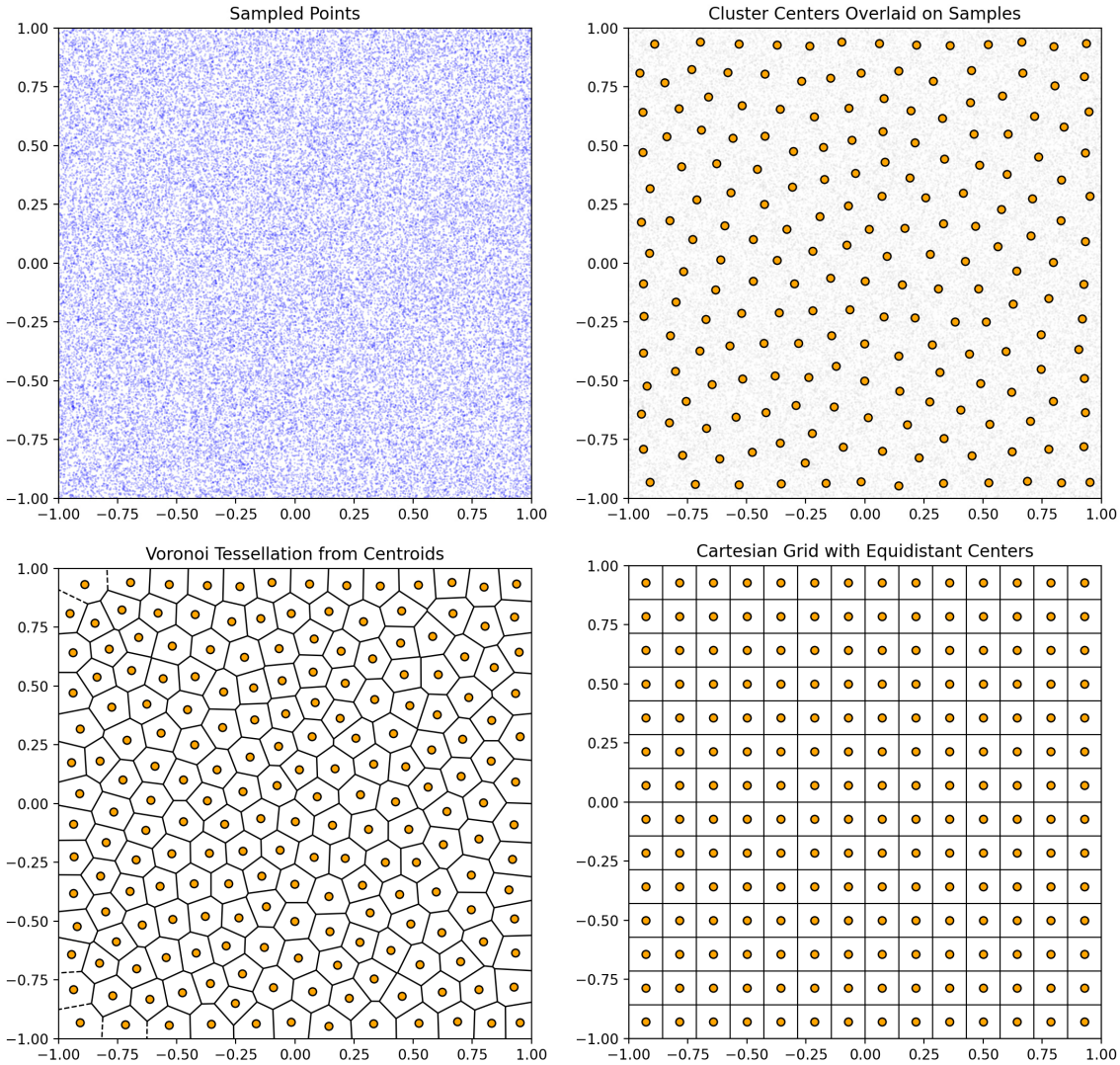


Figure 5.1: To create a Voronoi tessellation, the space is uniformly sampled. Then, using K-Means clustering and selecting the centres created, a smaller number of points is selected, which serve as the centroids for the tessellation. Compare in the second line the result with 200 centroids with a Cartesian grid with $14 \cdot 14$ cells.

This is different from using dimensionality reduction for a surprise score (or curiosity): There, the dimensionality reduction model is trained on the data of previous experiences. A reconstruction score or any other measure of how well the dimensionality reduction works is taken on the past and on new behaviour. The discrepancy between the quality of dimensionality reduction of past and new behaviour is then the surprise about this new behaviour. A high surprise score indicates new behaviour that should be explored further, see for example [EHL+20].

5.2.3 Improving the Offspring

In large parameter spaces, it is increasingly difficult to find good configurations just by applying random updates. Particularly in the area of neuroevolution, evolving a parametrisation of a potentially large neural network can require many generations and function evaluations before a good configuration is reached. This is particularly problematic in the reinforcement learning setup, as one stochastic function evaluation, rolling out an episode, is typically relatively expensive, at the very least because it requires multiple sequential forward passes.

One approach is to replace the random updates that create new offspring with a covariance matrix adaptation update [FTN+20] following the logic of CMA-ES [HO01]. Another popular idea is through *IsoLine* updates [VM18], originally designated as *Iso+LineDD*. After a parent x is selected, this strategy picks a second parent x' and creates a directional vector $x - x'$. Two variations on the first parent create the child: The *LineDD* element means moving along this directional vector randomly, but also distance-dependent, meaning the variance is dependent on the distance of $\|x - x'\|$. The *Iso* element stands for isotropic and means also moving a bit in a fully random direction, so isotropically. The variance of both influences is scaled with σ_{line} the *Line* σ hyperparameter and σ_{iso}, the *Iso* σ hyperparameter. The underlying motivation for this approach is the idea that solutions lie on a somewhat regular subspace, the elite hypervolume.

Another important approach for the parametrisation of large neural networks is Policy Gradient Assisted MAP-Elites (PGA-ME) [NC21]. It modifies the emitter, such that a portion, usually half, of the offspring are not created based on random variations of a parent, but rather by a gradient step based on reward information. To this end, global information about environment transitions is collected in an experience shared replay buffer and a continuously updated neural network that estimates the state-action value Q and serves as a critic function. This is implemented analogously to TD3 [FHM18], see also Subsection 3.1.5. TD3 is a good fit here because it is a policy-gradient method that uses deterministic actors, and typically, the policies evolved with quality-diversity algorithms are deterministic actors. It is off-policy, and by construction, learning in this context is off-policy, as experience is shared in the replay buffer.

A necessary modification to the TD3 algorithm is the additional creation and evolution of one shared *greedy* policy that selects the action a' at which the Q-function is evaluated to calculate the targets. This policy may learn with a distinct learning rate α_{greedy}. Each generation, the shared critics are updated for N_{critic} gradient steps and offspring is created by updating the copy of a given parent for N_{policy} gradient steps.

5.2.4 Measuring Quality-Diversity

Quality-diversity marks a shift in the underlying goal of the optimisation problem. Instead of searching for the one best solution, it is now a search for a population of solutions that is both diverse and consists of high-performing individuals. If the archive is fixed, two algorithms can be compared in their ability to fill the archive with high-performing solutions (see, for example [NC21]). This idea is represented with *coverage* and *QD-score* [PSS+15].

Definition 26 (Coverage and QD-score)**.** *Let* $\Phi = \dot{\bigcup}_{i=1,\dots,n} \Phi_i$ *be the partitioned behaviour space and* $\Pi_\Phi = \{\pi_i \mid \phi(\pi) \in \Phi_i\}_{i \in I \subset \{1,\dots,n\}}$ *be an archive of solutions. Let* $\eta_{min} = \min_\pi \eta(\pi)$ *be the lowest possible performance of any policy.*

Define $\frac{|I|}{n}$ *as the* coverage *and* $\sum_{i \in I}(\eta(\pi_i) - \eta_{min})$ *as the* QD-score *of the archive.*

This is a very useful metric to compare algorithmic changes that do not change the partition of the behaviour space, or more generally, the archive. It cannot serve as a testament to the abstract properties of quality and diversity of a population due to the circular argument of creating a population based on this notion of quality-diversity and then evaluating it on the same notion.

To gain a bit more neutral ground and specifically to compare two methods with different repertoires, a different shared reference is necessary. Novelty metrics inspired by the nearest-neighbour calculation of novelty search may be used for that [CD18] or, keeping with the paradigms of MAP-Elites, a reference repertoire may keep track of the learning process [PRB+22]. This then leads to a *projected QD-score*. An *accumulated* variant may observe the learning process:

Definition 27 (Accumulated and projected diversity metrics)**.** *Let* Π *be a repertoire of policies and* $\Phi = \dot{\bigcup}_{i=1,\dots,n} \Phi_i$ *be a partitioning of the behaviour space. Then*

$$\Pi_\Phi = \left\{ \underset{\pi \in \Pi, \phi(\pi) \in \Phi_i}{\arg\max} \ \eta(\pi) \right\}_{i \in I \subset \{1,\dots,n\}} \tag{5.5}$$

is the projected repertoire *and its coverage and QD-score are the* projected *coverage and QD-score of* Π.

The accumulated *projected coverage and QD-score are calculated with the set of policies* Π*, which is the collection of every policy added to the repertoire during the learning process.*

Another useful reference is the highest performance found in the repertoire. For example, when diversity in physical location is used to help solve a maze, the desired result is one proper solution to the maze. The success of any algorithm may be measured by how completely the maze has been explored, but also simply how successful the most successful individual is. An investigation of different behavioural descriptors using this method introduced the idea of proper alignment. That is, it is favourable to solve the problem when the behavioural descriptor is aligned with the actual goal. Alignment is the vague notion that the behavioural descriptor has some relationship to the problem at hand [PSS+15].

5.3 Intrinsic Value of Diversity

Diversity can be used as a tool to achieve certain goals. In novelty search, it can be used to drive exploration. These applications show the value of diversity. But it is also interesting to reflect on the intrinsic value of diversity, meaning the value of maintaining a diverse repertoire of solutions without a specific goal in mind. Can diversity that is not directed towards a certain goal still improve the outcome of the learning process? This question is somewhat opposed to the classic reading of diversity-encouraging algorithms that explicitly try to align the type of diversity that is encouraged with the type of problem that is encountered [PSS+15].

5.3.1 Populations with Diverse Behaviour

Differentiating reinforcement learning policies can drive the evolution of a population of agents that encompasses a wide variety of behaviours. Following the heuristic of evolutionary learning strategies, diversity is recognised as an important factor in developing creative

solutions that themselves may not perform well but serve as stepping stones for more sophisticated solutions [SBD+08; SL15].

Consider a 2-dimensional maze. An agent starts at the beginning of the maze. Its goal is to reach an endpoint. Setting the distance of the agent to the goal as the indicator for good performance, a simple reward function can be defined as the change in distance to the goal when making a move: Moving away from the goal incurs a negative reward signal, and moving towards the goal a positive reward signal. This setup obfuscates the natural strategy of solving a maze, to explore as much space as possible to find the correct path. While there are ways to encourage exploration in reinforcement learning algorithms, setups like this are also a metaphor and a simple application of the value of diversity: A population of agents that exhibits diverse behaviour implies its members take many different paths through the maze, which is effectively a strategy to cover as much space as possible. One agent may move away from the goal and stay far away from it. That agent is a low-performing solution and may get discarded if no value is placed on diversity. However, the correct way to the goal of a maze may exactly lead along such a path for some time before hitting the goal. Such a low-performing solution is an evolutionary stepping stone.

Algorithms like novelty search and MAP-Elites utilise ways to differentiate reinforcement learning policies as an integral part. This is typically done with custom, problem-specific characterisations of a policy. In the maze setting, this may be the last position of an agent. This way, a diverse population covers paths that end in different places in the maze, filling the whole space. The indicator of diversity, the last position of an agent, is directly linked to the desired outcome and the presupposed understanding of what makes a good stepping stone to a complete solution of the problem at hand.

But an algorithm is a tool to solve difficult problems, which makes this construction problematic. The indicator of diversity is chosen already with an understanding of the obstacles this problem poses and how to circumvent them. This severely limits the algorithm's ability to deal with unexpected challenges. Every new problem presents the challenge of first understanding the inherent obstacles to learning and selecting the appropriate behavioural descriptors. It is particularly inconsistent with the idea of "abandoning objectives" in favour of novelty as the title of a seminal paper on novelty search suggests [LS11b]. When the novelty marker has to be defined just as carefully as the reward feedback, no progress towards automation has been made.

With the previously introduced ideas on how to characterise and distinguish reinforcement learning policies based on their behaviour, two complementary opportunities arise: improving the algorithms, and a generalised *unsupervised* understanding of diversity relieves users from the burden to define the specific diversity that is necessary to avoid the inherent obstacles in a given problem. If there is some value in a diverse population, it may be found by selecting for diverse behaviour in a general sense, not only when selecting in a direct, problem-specific sense. Finding merit in these non-specific ways of differentiating policies shows their validity: The difference in these abstract numerical values actually corresponds to a meaningful difference in behaviour that can be practically leveraged.

5.3.2 The Benefits of Diversity

The intrinsic benefits of diversity are in themselves a complex and inspiring topic of research. The diversification of portfolios or crop diversity in agriculture mitigates risks that may be specific to one particular position or species. There, diversity is linked to reliability and

resilience. “Differentiation” is a standard approach in education, which presents students with diverse ways to learn about one subject. This recognises that different paths can lead to a similar goal, and it depends on the context, which is better.

For reinforcement learning applications, several beneficial properties of diverse populations have been recognised, both in a single-objective interpretation, where just one solution is selected from a population, and population-wide interpretations, where the population itself is the desired outcome of the learning process.

Probably the most important benefit is in exploration: The exploration-exploitation dilemma is a fundamental point of tension in any reinforcement learning algorithm [SB18], which is especially problematic in setups with sparse or deceptive rewards, see also Subsection 2.3.4. Exploration is particularly attractive for consideration as it is a hard and open problem for standard reinforcement learning algorithms, and it is easy to construct problems with exploration tasks, scale their difficulty, and measure an algorithm’s success.

The illumination of the solution space is another application that made the title “Illuminating search spaces by mapping elites” of the original publication that introduced the MAP-Elites algorithm [MC15]. Illumination designates the exploration of the solution space in a way that tracks several solutions encountered, which are different enough that each may be interesting to a user. This collection of candidate solutions is visualised or, more generally, arranged in an accessible way to better understand the presented problem through the lens of different approaches to solve it.

A third concept is robustness, or in other contexts, adaptability or flexibility. For reinforcement learning agents, it describes the capability of a controller to still perform in a changed environment. While conceptually very interesting, because it corresponds with an intuitive understanding of the role of diversity in evolution, it is much harder to track and quantify (see also Subsections 4.4.3 and 6.4.2).

5.3.3 Novelty as an Alternative Objective

Hard exploration setups (see Subsection 2.3.4) are particularly useful objects of research as they very clearly present the problem of local optima, a problem ubiquitous in reinforcement learning. To particularly research this problem, it makes sense to research hard exploration setups. While the term “hard exploration” is necessarily vague, the problem of local optima arises from flat or non-convex reward functions. Creating hard exploration setups can simply mean creating a sufficiently confusing reward function.

From this perspective, it is not surprising that methods trying to resolve this problem do so by adding information beyond reward to the learning process. Novelty search [LS11b] avoids a deceptive reward by completely ignoring the reward and replacing it with a search for new values encountered in the behavioural descriptor. Additionally, the original experiment uses mazes and replaces distance to the goal, a deceptive reward, with exploration of a helpful behavioural descriptor, the last physical position of the agent. While this achieves impressive results, it is tailored to the problem of deceptive mazes and not necessarily generalisable. To see an improvement, it requires both a deceptive reward signal and a low-dimensional and useful behavioural descriptor. A useful low-dimensional behavioural descriptor, however, already lends itself to the formulation of subgoals to circumvent deception in the reward landscape. This technique successfully solves deceptive maze environments [ŞB04].

Another problem with this approach is the missing gradient information. Reinforcement learning profits from the advances in deep learning, using backpropagation on neural networks.

Gradient information is necessary because the parameter space is large and evaluations are limited. Random perturbations typically used as mutation operators in evolutionary algorithms do not scale up to the complex problems that reinforcement learning tries to solve. Deep reinforcement learning algorithms utilise some differentiable loss function to parameterise fixed architectures of neural networks. This loss is typically based on transitions (s, a, r, s') from one state s to another s' given an action a and resulting in a reward r, and an evaluation $V(s)$ of a state or $Q(s, a)$ a state-action pair. A behavioural descriptor $\phi(\tau)$ is typically a function of a full trajectory τ. While this function ϕ may in certain circumstances be constructed as differentiable [FN21], in general it is not. It can be estimated using random variations [CMS+18], but that again requires an excessive evaluation budget.

In domains more complex than mazes, it also seems beneficial not to drop the information gained from the reward signal. The behaviour space can be too large, even after careful selection of the behavioural descriptor, such that too much computation time is wasted on exploring low-performing solutions. A study on a locomotion task, where a U-shaped trap obstructs a simulated humanoid walker, shows that a mixed objective between novelty and performance leads to better results than an objective that uses either by itself [CMS+18], even though the behaviour space is just set as the 2-dimensional last position, in line with the typical formulation when applied to mazes.

Novelty as an alternative objective harnesses the power of diversity, but it also does not describe proper diversity. It explicitly does not grant the same importance to a multitude of different, valid solutions, but rather rewards innovation and excellence that is still directed towards a certain goal. The next chapter asks the question of whether this relationship can be switched. Can a notion of generalised, unsupervised diversity that is not directed toward a specific goal still achieve such goals, only by harnessing the intrinsic value of diversity?

CHAPTER 6

Applied Diversity

Behavioural characterisation is the approach to understand a reinforcement learning policy not in terms of a function that maps from state to action space, but as a random element in either the state-action space or the space of trajectories therein. That point of view is fundamental to reinforcement learning. It is also a starting point for diversity-encouraging evolutionary strategies that tackle reinforcement learning problems.

Using evolutionary strategies instead of gradient-based optimisation can address the issue that many practical reinforcement learning problems are non-convex, that there may have multiple global and local optima. Diversity-encouraging evolutionary strategies typically work with specific behavioural descriptors that are aligned with a specific goal in mind, because a full, unfocused description is seen as either infeasible [VCM18] or as misguided [PSS16a].

For the better understanding and visualisation of cohorts of policies, generalised behavioural descriptors were shown to excel. These ideas can now also power quality-diversity algorithms to develop diverse populations that are diverse in a general sense without specific alignment. With the intrinsic value of diversity should come concrete, measurable impacts on the performance of algorithms when these behavioural descriptors are integrated.

Modifications to the MAP-Elites framework (see Section 5.2) are introduced in Section 6.1 that enable the use of the occupancy measure as the behavioural descriptor to encourage diversity in an unsupervised manner. The resulting novel [FG25] algorithm is explicitly described in Section 6.2. Section 6.3 evaluates this new method on benchmark problems and analyses its benefits for exploration and practical implications, including a novel problem setup [FG23] in Subsection 6.3.5. Section 6.4 sketches further use cases for this concept of unsupervised diversity.

6.1 Unsupervised MAP-Elites

Quality-diversity algorithms like novelty search with local competition [LS11a] emerged in part exactly to address the problem that exploration of the behaviour space is desired. This is attempted by fostering behavioural diversity when the behaviour space is too large to fully explore efficiently. Local competition then drives the search to only look in the solution space where higher performing individuals are found, reducing the space that is examined. MAP-Elites algorithms are often either used for illumination of the search space [MC15] or to develop a behaviour archive from which solutions can be selected that are useful because the goal or the environment has changed. However, they are also useful for exploration to solve hard exploration problems [FG23; FG25].

Specifically, when using quality-diversity methods to solve reinforcement learning problems, the requirement to create a problem-specific behavioural descriptor is bothersome. The quality aspect is directly defined as the overall performance of an agent, the underlying

optimisation problem in reinforcement learning. Defining a suitable behavioural descriptor to foster diversity, however, is not necessarily straightforward. It depends on the problem class. In some problem classes, such as mazes, a good approach applicable to the whole problem class exists. Other important problem classes, such as locomotion tasks in robotics, do not lend themselves to easy behavioural descriptors. An established descriptor for locomotion tasks is the amount of contact each leg of a robot has with the ground expressed as a ratio over one episode: It is implemented as a standard behavioural descriptor in QDax [CLB+24] and was demonstrated to find a population of gaits that can serve as an archive of behaviours that can be accessed in the case of damage to the robot [CCT+15].

Often, in the utilisation of diversity, a somewhat circular argument between diversity markers and their utility is made: The diversity of a population is, for example, defined by the usage of legs and then the utility of this diversity is measured by removing control over a leg. Here, the damage case is already anticipated in the formulation of the diversity marker. Similarly with mazes, the last physical position as a diversity marker already anticipates the deceptive nature of the problem setup and a path to overcome this problem. But part of the appeal of machine learning is that it may overcome problems that have not been anticipated and provided for during the design phase. This leads to the question of the intrinsic value of general diversity: *Is there value in a diverse population beyond the value that is gained in the direct alignment between the diversity marker and a secondary goal?*

The idea to use non-specific behavioural descriptors in quality-diversity has also been named *unsupervised quality-diversity* [CTC25]. The "supervision" aspect refers to the explicit, problem-specific design of a behavioural descriptor.

In some instances, the last state of an episode is taken as the behavioural descriptor, even if its utility is not as clear as with mazes. The argument is that the last state is a culmination of everything that has happened before, including the behaviour of an agent. In a way, defining this as a standard behavioural descriptor, regardless of the environment at hand, constitutes a non-technical implementation of unsupervised quality-diversity. An early instance of a technical advance of unsupervised quality-diversity was implemented in CVT-MAP-Elites [VCM18], see Subsection 5.2.1, which reformulated the way the behaviour space is partitioned away from a Cartesian grid to a centroidal Voronoi tessellation. This allowed higher-dimensional behaviour spaces, and one of the applications of this method is to assign a full episode, or rather a unified amount of equidistant samples of an episode, as the behavioural descriptor. This is done by taking a fixed number of snapshots along the episode and concatenating the data collected into one vector. Since no conscious choice on the behavioural descriptor is required, this is an unsupervised approach. Later approaches like AURORA [GC22b], which introduces the term unsupervised to this line of thinking, but also TAXONS [PLC+20], RUDA [GC22a], and STAX [Pao21] use learning algorithms to achieve dimension-reduction. The idea of a low-dimensional behavioural descriptor is maintained, but its design is now integrated in the learning loop.

Here, a different path to unsupervised quality-diversity is taken: The signature transform and the occupancy measure are both general behaviour descriptors in the sense that they use the full information available from observation and are problem agnostic, see Subsection 6.1.1. The MAP-Elites framework with the right extensions already can accommodate the signature transform as a behavioural descriptor. For the occupancy measure, this is not as trivial and adjustments and some shortcuts are necessary and described in Subsections 6.1.3

and 6.1.4. A complete description of the learning algorithm used in the experiments in this chapter is then summarised in Section 6.2.

6.1.1 General Behavioural Descriptors

From the reflection on visualisation in Section 4.1, distance measures with the occupancy measure [FG25] and through signature transform [FPG24; Pao21] have the potential to differentiate policies meaningfully.

A behavioural descriptor through the signature transform represents an approach that stays true to the episodic understanding of behaviour that is prevalent in quality-diversity techniques. It still sees the policy primarily through observed episodes as a time series. This episodic understanding is also the basis of the behavioural descriptors that take and concatenate snapshots or take just the last state. The signature transform, however, incorporates every observation, naturally resolves the problem of different-length episodes, and contains a conceptually satisfactory way to average multiple episodes to represent one policy, something very relevant when characterising episodes in Subsection 4.2.3.

A behavioural descriptor defined by the occupancy of a policy is a fundamental shift from the typical approach in MAP-Elites. It is somewhat reminiscent of some custom, aggregate behavioural descriptors that measure a ratio over a full episode, such as the behavioural descriptor that takes the percentage of time that the legs of a walker touch the ground (see also later in Subsection 6.3.2). With this idea, every observation is incorporated, episodes of different lengths do not pose a problem, and aggregation over several episodes is viable. The occupancy as a behavioural descriptor generalises this approach, incorporating not just one part of information in every observed moment but the full information.

They can be seen as general behavioural descriptors, as they do not pick and choose from the observed behaviour. Inaccuracies and information loss emerge from computation, discretisation, and representing a random element through observed samples. This relieves the user of MAP-Elites of designing an appropriate behavioural descriptor for the problem and protects them from carrying over their own preconceived notions about the problem into the learning algorithm.

6.1.2 Complexity of Behavioural Diversity

With the goal in mind to use for MAP-Elites some of the behavioural descriptors or behavioural distances that have been used for visualisation in Section 4.1, a short analysis of run time is in order. For this short reflection, assume a CVT archive, but the same argument translates to other setups as well.

Consider the different steps when adding a generation of N new candidates to the archive. For each individual, one episode is created, and the corresponding behavioural descriptors are calculated. Then, a distance matrix of size $N \times M$ needs to be calculated, where the behavioural descriptor of each new candidate solution is compared to each centroid to find the nearest one. Thus, significantly more behavioural distances are calculated than behavioural descriptors.

So, the behavioural distance calculated through signature transform and distance in signature space is much better suited for the MAP-Elites framework than dynamic time warping. To fit dynamic time warping into the MAP-Elites framework, the behavioural descriptor is trivially the episode itself. The distance between two behavioural descriptors is measured with dynamic time warping. Calculating this distance is expensive. This is what

makes dynamic time warping infeasible for MAP-Elites. Signature transform, on the other hand, is expensive as well, but a significantly smaller number of operations is necessary.

In general, for a distance measure to fit into the MAP-Elites framework, the calculation of the behavioural descriptor may be somewhat expensive, but the calculation of the pairwise distance may not. This also makes proper optimal transport infeasible for MAP-Elites.

6.1.3 Approximating the Occupancy Measure

To render the occupancy measure as a feasible option for a behavioural descriptor, simplifications are necessary. Properly solving the optimal transport between two point clouds is relatively expensive and, similarly to dynamic time warping, prohibitive to use in the MAP-Elites framework. First, assume the state-action space is bounded:

$$S \times A \subseteq [-1, 1]^n. \tag{6.1}$$

In general, this is not true, for example, when considering velocities in robotics applications. To address this problem, the state-action space is projected to be bounded with a normalising function. Here, the tangens hyperbolicus is used directly, resulting in bounds in $[-1, 1]$. This is not the optimal choice: In practice, a rolling normalisation and regular updates could improve the performance of this technique, but for the sake of clarity and brevity, this tweak is omitted. Then, this space is discretised into cells:

$$\dot{\bigcup}_{i=1,\dots,M} \Phi_i = [-1, 1]^n. \tag{6.2}$$

To keep the number of bins reasonably small, a centroidal Voronoi tessellation is used for discretising the transformed state-action space into cells as in Subsection 5.2.1. Sampling a trajectory $\tau \sim \pi$ the estimated occupancy measure $\phi_{\text{occ}}(\tau) \in [0, 1]^M$ is defined as the visitation frequency of each cell:

$$\phi_{\text{occ}}(\tau)_i = \frac{|\{(s_i, a_i) \in \tau \cap \Phi_i\}|}{|\tau|}, \tag{6.3}$$

see Figure 6.1 for a sketch of this idea. By construction $\sum_{i=1,\dots,M} \phi_{\text{occ}}(\tau)_i = 1$, so the image

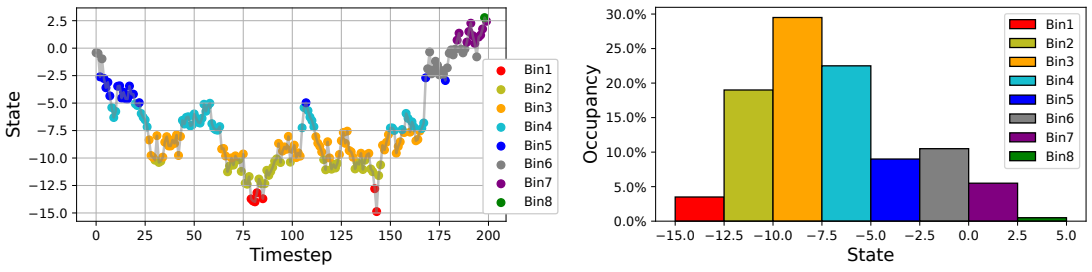


Figure 6.1: Discretise the state(-action) space, here rendered in a sketch as 1-dimensional. With this discretisation, an episode can be translated to a discrete distribution that approximates the occupancy measure that pertains to that policy.

of ϕ is actually the $M-1$ simplex:

$$\phi_{\text{occ}} : (S \times A)^{\mathbb{N}} \to \Delta^{M-1} = \left\{ x \in [0,1]^M \mid \sum_{i=1,\dots,M} x_i = 1 \right\}. \tag{6.4}$$

Since each element in the behaviour space $\Phi = \Delta^{M-1}$ actually represents a discrete distribution, a proper distance measure is more difficult to define than for traditional behavioural descriptors: The L^1 distance is a simple approximation: It is equivalent to the transport cost, with the simplification that every point has the same transport cost to any other point.

This constructs only the behavioural descriptor. To finish the MAP-Elites setup, an archive has to be set up which governs the selection of the population. To that end, a second centroidal Voronoi tessellation is created in the behaviour space Δ^{M-1}. To create this tessellation, a uniform sampling of this particular space is in order. Sampling the Dirichlet distribution with the concentration parameter $\alpha \equiv 1$ is used to achieve that.

6.1.4 Sinkhorn Distance

As the experiments will show, the L_1 distance is a surprisingly effective shortcut to work with the occupancy measure. A more theoretically sound approach is the use of the Sinkhorn distance [Cut13] as a more abstract, but also more truthful representation of proper optimal transport. The Sinkhorn distance is an approximation of the optimal transport distance that can be estimated faster than optimal transport. A wide variety of machine learning applications use this idea, for example, to measure distances between crystallographic textures [IML+24].

Let s, d, C be supply, demand, and cost of an optimal transport problem. Designate $\langle \cdot, \cdot \rangle_F$ as the Frobenius inner product. Then the optimal transport problem as defined in Subsection 4.1.5 can be understood as the minimising value

$$d_C(s,d) = \min_{Z \in U(s,d)} \langle Z, C \rangle_F, \qquad \text{where} \tag{6.5}$$

$$U(s,d) = \{ Z \in \mathbb{R}_+^{n \times n} \mid Z \cdot s = \mathbb{1}_N, Z^T \cdot d = \mathbb{1}_M \}. \tag{6.6}$$

The Sinkhorn distance can be understood as an entropy-regularisation of that minimisation problem. The entropy h is defined as $h(x) = -\sum_i x_i \log x_i$. The minimisation is constrained to the transport plans Z with higher entropy.

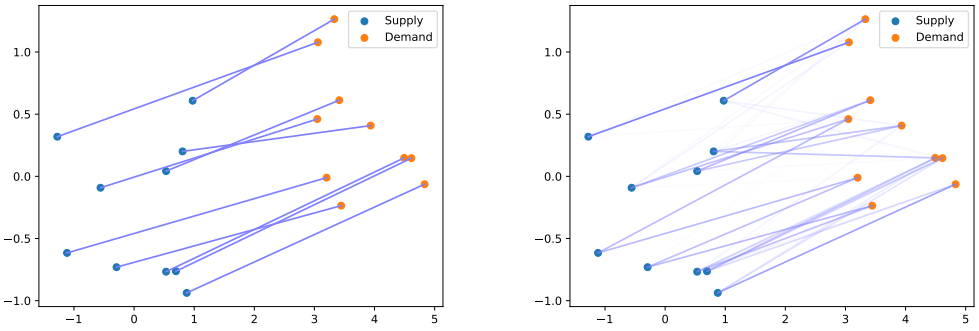
Definition 28 (Sinkhorn distance)**.** *Let s, d, C be supply, demand, and cost of an optimal transport problem, and $\alpha \geq 0$. Then the* Sinkhorn distance *of s and d is*

$$d_{C,\alpha}(s,d) = \min_{Z \in U_\alpha(s,d)} \langle Z, C \rangle_F \tag{6.7}$$

where

$$U_\alpha(s,d) = \{ Z \in U(s,d) \mid h(Z) \geq h(s) + h(d) - \alpha. \tag{6.8}$$

For a Lagrange multiplier $\lambda > 0$, the transport plan Z that minimises in Equation (6.7) corresponds to a transport plan Z^λ that minimises an entropy-regularised minimisation



(a) Optimal transport plan **(b)** Sinkhorn transport plan

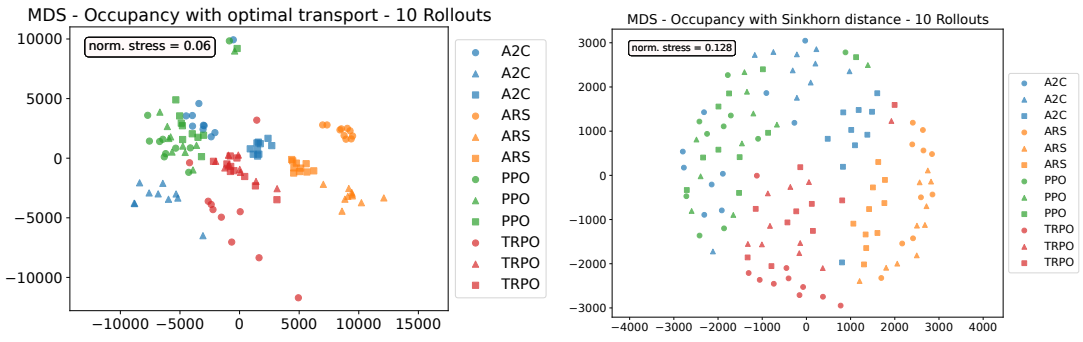
Figure 6.2: Sketch to visualise the Sinkhorn distance in comparison to optimal transport distance. The respective transport plan that connects supply and demand (i.e., the two sample sets) is visualised. A darker colour indicates a higher weight on that edge. All weights are positive. Edges exclusively connect one supply node with one demand node. For every node, all connected edge weights sum up to 1. The transport cost is the sum of the edge weights multiplied by the distance that the edge represents.

problem resulting in a *dual-Sinkhorn divergence*:

$$Z^{\lambda} = \underset{Z \in U(s,d)}{\arg\min} \langle Z, C \rangle_F - \frac{1}{\lambda} h(Z) \tag{6.9}$$

The Sinkhorn transport plans result in a more equitable mapping of supply to demand. While in optimal transport, most connections either fully exhaust the supply or the demand node, the Sinkhorn transport plans typically connect one supply point with all demand points, only favouring those connections that reduce cost with larger weights. Figure 6.2 exemplifies how two concrete transport plans differ.

With this additional distance measure, the visualisation of Chapter 4 can be repeated: Figure 6.3 shows a side-by-side comparison of the visualisation of 12 CartPole agents with



(a) Optimal transport **(b)** Sinkhorn

Figure 6.3: Comparison of the visualisation of episodes in the CartPole environment with aggregation of 10 episodes using either optimal transport distance or Sinkhorn distance of the occupancy measure. The pairwise distances are visualised with multidimensional scaling.

Table 6.1: How much runtime do different distance measures require? Measured here is the time it takes to calculate 1000 distances between observed episodes in CartPole.

Metric	s/1000 Calculations
Snap. 100	0.002
Last Pos.	0.002
Sgnt. 3	0.002
OT	47.18
DTW	87.77
Sinkhorn	10.21

10 episodes aggregated with optimal transport that was previously shown in Figure 4.10 and the same visualisation repeated but using the Sinkhorn distance. The relationship between the different data points is, in parts, still recognisable: Observations of the PPO agents and one A2C agent overlap, the other two A2C agents are close by but somewhat distinct. Observations from the ARS agents are well separated from the other observations, and observations from the TRPO agents are separated but not as clearly. However, the distances are smoothed out significantly, and nuances are lost. For example, optimal transport can distinguish the different ARS agents, and the Sinkhorn distance cannot.

The Sinkhorn distance can be calculated much more efficiently than optimal transport with a complexity of $O(n \log n)$ instead of $O(n^3 \log n)$, where n is the dimension of supply and demand. In practice, it is solved with an iterative algorithm. To utilise it in the context of MAP-Elites, it was still necessary to reimplement the algorithm to execute it in parallel with GPU-acceleration. Table 6.1 shows a comparison of observed runtimes for the different distance measures in the CartPole environment without GPU-acceleration. The distance operations that do not require solving an optimisation problem are very fast compared to optimal transport, Sinkhorn, and dynamic time warping. That explains the difficulty of working with these complex distance measures in an algorithm that requires repeatedly calculating these distances.

6.2 PGA-ME with Occupancy

Using the occupancy measure as the behaviour descriptor is a novel idea introduced here. That warrants a detailed exposure of the algorithm. The algorithm is based on the MAP-Elites (ME) framework [MC15] with a centroidal Voronoi tessellation to define the archive [VCM18] and policy gradient assistance (PGA) to improve the performance of the suggested new solutions during learning [NC21].

This section mostly repeats, clarifies, and summarises details that were introduced in Section 6.1, in Section 5.2, and Subsection 3.1.5. There are three parts to MAP-Elites: Setup, initialisation, and the evolutionary loop. The standard evolutionary loop of MAP-Elites can be further segmented into three main steps after initialisation: Reproduction, evaluation, and selection. An algorithmic description in words is given in Algorithm 6 that includes the extensions to MAP-Elites used for experiments in this chapter. Every statement in Algorithm 6 is explained in this section.

The *setup* defines how all the elements of the generic algorithm connect to address a given problem:

Algorithm 6 Pseudocode for PGA-ME with occupancy

Set up an architecture for policies and critics.
Set up an empty archive.
Set up centroids for discretising the observed transformed state-action pairs.
Set up centroids that decide which policies are competing with each other.
Set up a replay buffer.
Initialise critic weights and greedy actor weights.
Create an initial population.
For every member of the initial population, observe its behaviour with occupancy.
Add every observed transition to the shared replay buffer.
Add every member of the initial population to the archive based on its behaviour.
for each generation **do**
    Train the policy gradient assistance.
    Create offspring.
    For every member of the offspring, observe its behaviour with occupancy.
    Add every observed transition to the shared replay buffer.
    Add every member of the offspring to the archive based on its behaviour.

Set up an architecture for policies and critics

Genotypes and a mapping between genotypes and phenotypes have to be defined. Here, this means fixing an architecture for a simple feed-forward artificial neural net with ReLU activation functions and two hidden layers whose size is set depending on the problem, see Appendix A.2. The input and output layers are shaped such that the resulting function can serve as a policy $\pi : S \to A$. The weights $\theta \in \mathbb{R}^n$ of the neural net are the genotype, the resulting function π^θ its phenotype:

$$\text{Define } \pi^\theta(.) \leftarrow \pi(\theta, .) : S \to A \tag{6.10}$$

Set up an empty archive

The empty archive $X = \mathbb{R}^{M\times(n+1)}$ is best interpreted as an array with M entries initialised as $\{X_i = (\mathbb{0}_n, -\infty)\}_{1,\dots,M}$. In that array, the position at index i represents an individual of the population associated with the i-th centroid. The parameterisation of a solution is the first n elements, the observed performance is the $(n+1)$-th element.

$$\text{Define } X \in \mathbb{R}^{M\times(n+1)} \tag{6.11}$$

$$X_i = (0, \dots, 0, -\infty) \in \mathbb{R}^{n+1} \tag{6.12}$$

Set up centroids for state-action pairs

To set up the behavioural descriptor, M^{occ} centroids $\lambda^{\text{occ}}_{i=1,\dots,M^{\text{occ}}} \in [-1, 1]^{|S\times A|}$ are created by uniformly sampling the space $n^{\text{sample}}_{\text{cvt}}$ times and applying k-means clustering to find M^{occ}

cluster centres. The cluster centres are the centroids for discretising the observed transformed state-action pairs.

$$Y = \{y_i \in \mathbb{R}^{|S\times A|} \mid y_{i,j} \sim \mathcal{U}_{[-1,1]}\}_{i=1,\dots,n_{\text{cvt}}^{\text{sample}}} \tag{6.13}$$

$$\{\lambda_i^{\text{occ}}\}_{i=1,\dots,M^{\text{occ}}} = \text{KMeans}(Y, M^{\text{occ}}) \tag{6.14}$$

Set up centroids for local competition

Also M centroids $\lambda_{i=1,\dots,M} \in \Phi = \Delta^{M^{\text{occ}}-1}$ are created by uniformly sampling the space $n_{\text{cvt}}^{\text{sample}}$ times and applying k-means clustering to find M cluster centres. The cluster centres are the centroids that are relevant for deciding which policies are competing with each other.

$$Y = \{y_i \in \Delta^{M^{\text{occ}}-1} \mid y_i \sim \text{Dir}(\mathbb{1}_{M^{\text{occ}}})\}_{i=1,\dots,n_{\text{cvt}}^{\text{sample}}} \tag{6.15}$$

$$\{\lambda_i\}_{i=1,\dots,M} = \text{KMeans}(Y, M) \tag{6.16}$$

Set up a replay buffer

A shared replay buffer is set up as a queue that holds N_{buff} transitions. That is a list which can hold up to N_{buff} elements. If more elements are added to the list, elements are removed from the list, until it holds exactly the maximum number of elements. Elements are removed "first-in-first-out".

$$\text{ReplayBuffer} \leftarrow \text{Queue}(N_{\text{buff}}) \tag{6.17}$$

Initialise critic weights and greedy actor weights

For the policy gradient updates, shared critic weights ψ_1, ψ_2, and greedy actor weights θ_{greedy} are initialised by sampling the normal distribution $\mathcal{N}(0,1)$ for every entry alongside targets ψ_1', ψ_2', θ'_{greedy} by copying the shared critic and greedy actor weights. The weights parameterise deep neural networks with the same architecture as the policies, only the input and output layers differ for the critics, that are functions $S \times A \to \mathbb{R}$, while the actor is a function $S \to A$. Due to the different input and output layers of the critics, they potentially have a different number $\tilde{n}$ of weights compared to the policy architecture.

$$\text{Define } Q^{\psi}(\cdot,\cdot) \leftarrow Q(\psi,\cdot,\cdot) : S \times A \to \mathbb{R} \tag{6.18}$$

$$\psi_i = (\psi_{i,j})_{j=1,\dots,\tilde{n}} \qquad \psi_{i,j} \sim \mathcal{N}(0,1),\ i = 1,2 \tag{6.19}$$

$$\theta_{\text{greedy}} = (\theta_{\text{greedy},j})_{j=1,\dots,n} \qquad \theta_{\text{greedy},j} \sim \mathcal{N}(0,1) \tag{6.20}$$

$$\psi_i' = \psi_i \qquad \text{for } i = 1,2 \tag{6.21}$$

$$\theta'_{\text{greedy}} = \theta_{\text{greedy}} \tag{6.22}$$

Repeated operations

Now, for the initialisation and evolutionary loop, two repeated operations need to be clarified. They are motivated and further explained in Subsection 6.1.3.

Definition 29 (Observing the behaviour with occupancy)**.** *Let $\pi : \mathbb{R}^n \times S \to A$ be a fixed architecture for a policy in a Markov decision process $\mathcal{M}$. Write $\pi^{\theta}(s) = \pi(\theta, s)$. Let $\{\lambda_i^{occ}\}_{i=1,\dots,M^{occ}}$ be points (centroids) in $[-1,1]^{|S\times A|}$.*

Call observing the behaviour *of an individual $\theta \in \mathbb{R}^n$, the following process that finds the behavioural descriptor for the occupancy measure $\phi_{occ}(\pi^\theta)$ and performance $\eta(\pi^\theta)$ of θ:*

First, sample an episode

$$\tau = (s_j, a_j, r_j)_{1,\dots,m} \sim \pi^\theta. \tag{6.23}$$

The performance is the undiscounted accumulated reward of the sampled episode

$$\eta(\pi^\theta) = \sum_{j=1}^{m} r_j. \tag{6.24}$$

The observations are transformed to $[-1, 1]$ by setting

$$\tilde{s}_j = \tanh(s_j) \tag{6.25}$$

$$\tilde{a}_j = \tanh(a_j). \tag{6.26}$$

For each centroid λ_i^{occ} in the state-action space, count how many state-action pairs are closest to that centroid. If there is a tie, the state-action pair is counted towards the centroid with the lower index number. The behavioural descriptor is calculated by dividing this vector of integers by the total number of state-action pairs observed:

$$(\phi_{occ}(\pi^\theta))_i = \frac{1}{m}\left|\left\{j \in \{1, \dots, m\} \mid j = \underset{k=1,\dots,m}{\arg\min} \|(\tilde{s}_k, \tilde{a}_k) - \lambda_i\|\right\}\right| \tag{6.27}$$

Definition 30 (Adding to the archive)**.** *Let $\pi : \mathbb{R}^n \times S \to A$ be a fixed architecture for a policy in a Markov decision process $\mathcal{M}$. Write $\pi^\theta(s) = \pi(\theta, s)$. Let $\lambda_{j=1,\dots,M}$ be points (centroids) in Φ, the behaviour space. Let d be a distance measure in Φ. Let $X \in \mathbb{R}^{M\times(n+1)}$ be an appropriate archive.*

To add θ to the archive *based on its behaviour $\phi_{occ}(\pi^\theta)$ and performance $\eta(\pi^\theta)$, find the index i of the centroid $\lambda_{j=1,\dots,M}$ that is closest to the behavioural descriptor $\phi(\pi^\theta)$ of the individual according to the distance measure d:*

$$i = \underset{j=1,\dots,M}{\arg\min} \left[d\left(\phi_{occ}(\pi^\theta), \lambda_{j=1,\dots,M}\right)\right] \tag{6.28}$$

If there is a tie, the index i with the lowest value is selected.

Then, if the observed performance $\eta(\pi^\theta)$ is bigger or equal to the stored performance of the i-th element of the archive $X_{i,(n+1)}$, the new solution θ is stored in the archive, that is

$$\text{if } X_{i,n+1} \leq \eta(\pi^\theta) : X_i \leftarrow (\theta, \eta(\pi^\theta)) \tag{6.29}$$

Create, observe, and add the initial population

Initialisation starts with creating a population $\{\theta_i\}_{i=1,\dots,N_{\text{pop}}}$ by sampling the normal distribution $\mathcal{N}(0, 1)$ for every entry.

$$\theta_i = (\theta_{i,j})_{j=1,\dots,n} \qquad \theta_{i,j} \sim \mathcal{N}(0, 1),\ i = 1, \dots, N_{\text{pop}} \tag{6.30}$$

Then their behaviour is observed following Definition 29 and they are added to the archive following Definition 30. The transitions of the observed episode are added to the replay buffer.

$$\text{for } i = 1, \dots, N_{\text{pop}} : \tag{6.31}$$

$$\phi_{\text{occ}}(\pi^{\theta_i}), \eta(\pi^{\theta_i}), \tau \leftarrow \text{ObserveBehaviour}(\theta_i) \tag{6.32}$$

$$\text{Add } \tau \text{ to Buffer} \tag{6.33}$$

$$X \leftarrow \text{AddToArchive}(X, \theta_i, \phi_{\text{occ}}(\pi^{\theta_i}), \eta(\pi^{\theta_i})) \tag{6.34}$$

Train the policy gradient assistance

Now, the evolutionary loop starts. This is managed based on the Policy Gradient Assisted MAP-Elites framework (PGA-ME) [NC21]: Before reproduction, the policy gradient assistance is trained based on the shared replay buffer and with the shared greedy actor. This is done mirroring the update step in TD3 (Algorithm 2):

for $j = 1, \dots, N_{\text{critic}}$ **do**

Sample batch of N_{batch} transitions $\{(s, a, r, s')\}$ from the replay buffer.

Let $a' = \left[\pi^{\theta'_{\text{greedy}}}(s) + [\varepsilon]_c\right]_{A_s}$ with $\varepsilon \sim \mathcal{N}(0, \sigma)$ where $[x]_c$ indicates clipping such that $x \in [-c, c]$ and $[x]_{A_s}$ clipping such that $x \in A_s$.

Let $y \leftarrow r + \gamma[s \notin S_{\text{term}}] \min_{i=1,2} Q^{\psi'_i}(s', a')$, where $[P]$ is 1 if P is true and 0 otherwise.

Update ψ_i with loss $L(\psi_i) = (Q^{\psi_i}(s, a) - y)^2$ and learning rate α_{critic} for $i = 1, 2$.

if $(j{+}{+}) \bmod n_{\text{delay}} = 0$ **then**

Update θ_{greedy} with $\nabla_a Q^{\psi_1}(s, a)_{|a=\pi^{\theta_{\text{greedy}}}(s)} \nabla_{\theta_{\text{greedy}}} \pi^{\theta_{\text{greedy}}}(s)$ and learning rate α_{policy}.

Soft update of targets: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$, $\psi'_i \leftarrow \tau\psi_i + (1 - \tau)\psi'_i$ for $i = 1, 2$.

Create, observe, and add offspring

Then, for reproduction, offspring are created by copying a uniform selection with replacement of individuals that are in the archive. Each copy is then updated with a 50% chance for an IsoLine update and 50% chance for a policy gradient update. An IsoLine update on an individual θ is done by randomly sampling another member of the population $\tilde{\theta}$. The update is set as

$$\theta' = \theta + \mathbb{1}_n \mathcal{N}(0, \sigma_{\text{iso}}) + (\tilde{\theta} - \theta)\mathcal{N}(0, \sigma_{\text{line}}) \tag{6.35}$$

A policy gradient update is done by uniformly sampling N_{policy} times a batch of size N_{batch} transitions from the replay buffer and applying a gradient step with

$$\nabla_a Q^{\psi_1}(s, a)_{|a=\pi^{\theta}(s)} \nabla_{\theta} \pi^{\theta}(s) \tag{6.36}$$

and learning rate α_{policy}.

This way, the offspring is created, and the reproduction step is finished. Next, for each individual in the offspring, an episode is rolled out, its behavioural descriptor is calculated (evaluation), and it is added to the archive (selection). The loop is repeated until a predefined number of loops have been completed. The value set in the following experiments represents

a trade-off between giving the algorithm enough time to find good solutions and controlling computational effort.

Hyperparameters

This full process connects multiple different ideas and approaches for an effective exploration of different strategies. Each of these separate ideas comes with some "magic numbers": Hyperparameters that can be tuned to a specific task, but within reasonable bounds should neither much hinder nor accelerate the process. See Appendix A.2 for a full summary of the choices.

6.3 The Impact of the Behavioural Descriptor in Exploration

The impact of the choice of the behavioural descriptor can be analysed in an experimental manner (see also [FG25]). To this end, on a fixed set of environments, architectures of candidate solutions, and a fixed algorithmic framework, behavioural descriptors are swapped and the impact of that exchange documented.

6.3.1 Behavioural Descriptors

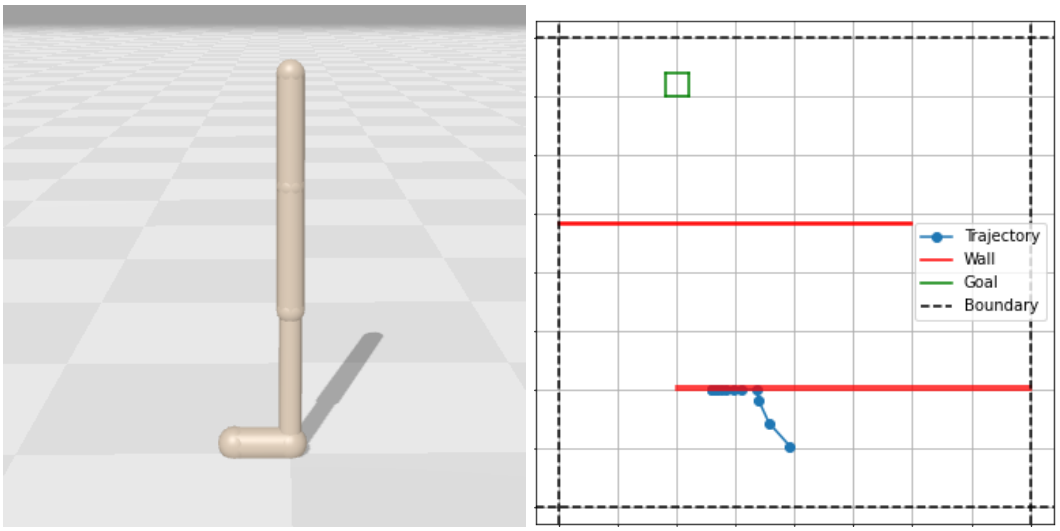
The behavioural descriptors considered for this study on unsupervised quality-diversity are for established "naive" descriptors, the *Last State-Action* descriptor, and the *20 Snapshots* descriptor, which selects 20 equidistant samples of an episode, as in Definition 19. They are compared to the behavioural descriptors inspired by the reflection on distance in the behavioural space in Section 4.1 and developed in Section 6.1: *Occupancy L1*, using the occupancy measure as behavioural descriptor and the L_1 metric, *Occupancy Sinkhorn*, using the occupancy measure as behavioural descriptor and the Sinkhorn distance, *Signature Depth* k, using the signature transform with depth k as behavioural descriptor.

As an upper baseline for the utility of diversity, consider also the custom behavioural descriptors that is the default for each environment in QDax: *Custom BD*. As a lower baseline for the utility of diversity within the confines of the algorithm, let *Random BD*, the random behaviour descriptor ϕ_{rand} be a descriptor that returns a uniformly sampled number in $[0, 1]$ independently sampled from the observed episode. The behaviour space for this descriptor is just the interval $[0, 1]$. This behavioural descriptor can be fit in the same evolutionary loop, but should in no way encourage behavioural diversity further than what the MAP-Elites setup itself delivers. Evolutionary niches still exist, but the allocation of new policies to these niches is completely random.

6.3.2 Benchmark Problems

For an analysis of the capability of unsupervised quality-diversity to explore the solution space, hard exploration environments are the right choice. The environments PointMaze, AntMaze, and AntTrap implemented in QDax [CLB+24] are used here. The Walker2D environment, also implemented in QDax, represents a more typical reinforcement learning problem.

For a typical reinforcement learning problem, consider the Walker2D environment, a bipedal walker, implemented as in Brax [FFR+21]. It has a 17-dimensional state-space, containing the absolute position of the torso, the angles of its six joints, and the derivatives of these values, except the forward momentum of the walker, which is encoded in the reward. Its action space is 6-dimensional and bounded by $(-1, 1)$. Each action dimension is the actuation of one of the 6 joints. The reward function is defined as a weighted sum of three



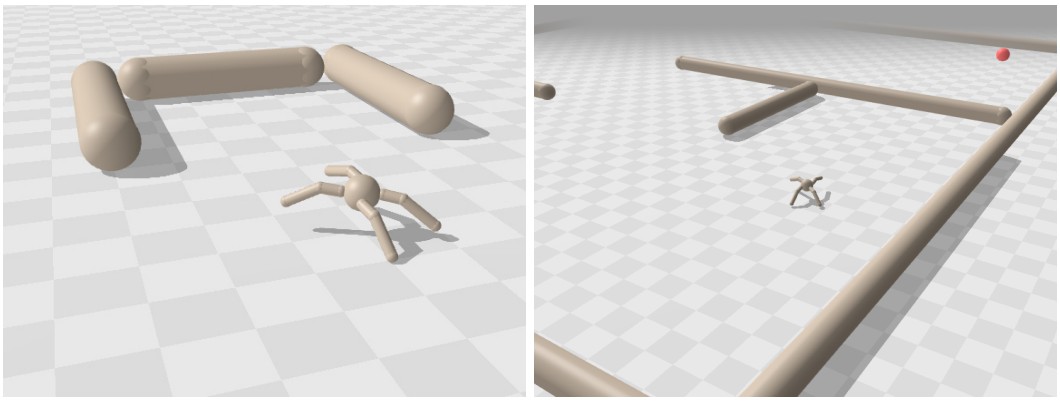
(a) Walker2D **(b)** PointMaze

Figure 6.4: A depiction of the bipedal walker *(Walker2D)* and a simple maze environment *(PointMaze)* in local optima observed when training with PPO: The Walker stands still, the point is drawn in a straight line to the goal, disregarding the obstacle.

components: A positive, flat reward for being healthy, which means not falling on the ground, a dynamic reward for moving forward in the desired direction, and a dynamic negative reward for actuation to encourage energy efficiency.

While the healthy reward is flat, neither the forward nor the actuation cost is flat. The problem cannot immediately be identified as convex or non-convex. It is still somewhat straightforward and would not be identified as a hard exploration problem. However, there are already at least two local optima that a learning algorithm can get stuck in: Falling or jumping straight forward incurs a modest reward, but ends the episode quickly. It can be a useful stepping stone in learning to walk if the agent finds a way to catch the body after falling. Also, standing still is a mode that collects the healthy reward while incurring no cost from using the actuators (see Figure 6.4(a)). This is not a useful stepping stone, as each movement will incur a penalty from the control cost, so this configuration is encouraged to stay as still as possible. Clearly, exploration capabilities are necessary in every reinforcement learning setup, not just in hard exploration problems. The walker task may have even more local optima: Maybe different types of gaits, like walking and running, are not continuous evolutions of one another but different skills. Walking would then be a local optimum that is separate from running. A method that is able to find, but also to leave a local optimum, is then superior to a learning method that stays in an optimum.

A typical problem in diversity-driven reinforcement learning problems is mazes [LS11b; PSS16b]. As a first example, consider the PointMaze environment. This setup, implemented in QDax [CLB+24], poses the task to move a particle from a starting position to a goal. Between the particle and the goal are two walls that obstruct the movement of the particle with a shifted opening in each. Its state space is the 2-dimensional, representing the position of the particle. The action space is 2-dimensional, representing a vector of movement for the



(a) AntTrap **(b)** AntMaze

Figure 6.5: A depiction of the quadruped ant, obstructed by a U-shaped trap *(AntTrap)* and by a maze *(AntMaze)*. These obstacles create a deceptive reward signal.

particle. The reward signal is defined as the negative distance of the particle from the goal. When reaching the goal, the episode ends, and no further penalties are incurred.

In this setup, the reward signal draws the agent to move in a straight line towards the goal (see Figure 6.4(b)). However, a global solution would take the particle as fast as possible towards the two openings before moving towards the goal. This constitutes a deceptive reward. Although this problem seems trivial, in this setup and with the usual tools of reinforcement learning, it is not. It is a hard exploration problem.

To demonstrate and test that capability in a more challenging environment, with AntTrap and AntMaze, hard exploration problems are also implemented in the benchmark suite QDax (see Figure 6.5). Here, the Ant environment is modified to include obstacles, which creates a combination of a maze-like environment with a locomotion task. The AntTrap environment is a unidirectional walking task, and a U-shaped obstacle is placed in the path where the reward function points. In the AntMaze environment, the reward signal that is previously given by forward movement towards one direction is replaced with a negative signal that is the distance to a goal, which is rendered as a red sphere in Figure 6.5(b). The closer the agent is to the goal, the lower the negative reward. The locomotion task that is contained in these environments requires a significantly larger state and action dimension than a simple particle in a maze. This makes this environment both challenging to reinforcement learning algorithms due to its deceptive reward structure, but also challenging to evolutionary algorithms that foster diversity and typically operate without gradient information.

There are several possible metrics that can be considered for the analysis of the results. The transformation back to a single-objective task is an obvious one: After an evolutionary optimisation run, pick the best individual of the population and measure its *performance*. In the case of these exploration tasks, there is some conceptual obstacle to overcome. So not only the performance of this best individual should be measured, but also if this individual managed to overcome these obstacles, or if it has *solved* the task.

Since this analysis is rooted in the ideas of quality-diversity, classic metrics to consider would be *coverage* and *QD-score* of the archive, see Subsection 5.2.4. These are the percentage of niches filled in the archive and the sum of performance of the individuals in those niches after this value has been incremented by a constant number, such that it is at least zero. This

Table 6.2: Dimensions of four concrete environments and the dimensions of the signature transform of episodes in those environments

	Episode Length	State dim.	Action dim.	Sgnt. 2 dim.	Sgnt. 3 dim.
Point-Maze	200	2	2	21	85
Walker2D	1000	17	6	553	12720
Ant-Trap	1000	95	8	10713	1103440
Ant-Maze	3000	101	8	11991	1307020

is done to ensure that a niche filled always leads to a higher QD-score than an empty niche. However, these metrics are not useful when comparing different strategies to create and work with niches, for example, as a result of changing the behavioural descriptors. Consider, for example, the random behavioural descriptor ϕ_{random}. Filling every niche, resulting in a coverage score of 100, is trivial, but that says nothing about the diversity or the algorithm's capability to explore.

So, for this task, also consider a *projected* and *accumulated projected* coverage and QD-score (see Definition 27). In the case of PointMaze, AntMaze, and AntTrap, the reference archive is set in the behaviour space of the last physical location of the simulated robot as seen from above, in the plane. In the case of the Walker2D, the behaviour space is the ratio of each foot touching the ground. Observing the accumulated projected scores over the learning process, these metrics are strictly increasing. The projected, not accumulated, scores may also decrease over time. They offer a more truthful insight into how population diversity behaves over the course of the optimisation run.

6.3.3 Runtime and Memory

Considering different ways to describe the behaviour of a policy in an algorithm also requires an analysis of computational issues with the new approaches. In practice, two bottlenecks emerged: Runtime and memory usage. Keeping in mind that computing power continues to increase, it is less interesting to see what works now, and rather what scales properly with increasing computing power. [Moo65; Sut19].

Using the signature transform to describe episodes on a high-dimensional state-action space, there are memory issues. The dimension of the signature of depth d and a state-action dimension of N is $\sum_{i=0}^{d} N^d$. Table 6.2 shows the impact of that for the four environments considered.

A standard number representation in Python (a numpy float64) takes up 8 bytes. That means that one data point with a million parameters uses approximately 8 megabytes. Storing 1000 such data points already requires 8 gigabytes of RAM, with an additional 8 gigabytes required by the behaviour of the next generation. Running the MAP-Elites algorithm with signature depth 3 in an Ant environment requires storing such data points both for the centroids and for the offspring of each new generation. That adds up to at least 16 gigabytes just for storing the behavioural descriptors, which is half of the RAM that state-of-the-art GPUs have at this time. With the additional requirements for running the simulation and everything else in the optimisation loop, for both Ant environments, only the signature transform of depth 2 could be run. Even factoring in increasing computing power, the exponential scaling of the dimensionality of the transformed data with the

depth of the signature transform will make it difficult to increase depth for applications in population-based approaches.

The second issue, runtime, is shown in Table 6.3. The table shows the runtime of advancing the algorithm for one generation in seconds. The lower bound for varying behavioural descriptors is the *Random BD* or the *Last State-Action*, as there is almost no calculation necessary for these behavioural descriptors. Runtimes that exceed these numbers can be explained by the computational effort connected with the behavioural descriptor. No effort can be assigned to using a proper optimal transport distance, as not even one generation can be completed. Table 6.3 shows that the behavioural descriptors suggested here have just a small impact on computing time. However, the exponential scaling problem with signature transform can be seen in the jump of effort from PointMaze to Walker2D: In PointMaze, the increase in runtime switching from the trivial *random* behavioural descriptor to the signature was about 0.15 seconds or 10%. In the Walker2D environment, the increase was about 11.5 seconds or almost 400%. This difference can be explained by the vastly different dimensions of the transformed episodes (85 to 12720, see Table 6.2) and the connected computational effort to calculate their values.

6.3.4 Benchmark Evaluation

Measuring the impact of the choice of behavioural descriptor on the performance of the best individual in each environment, there are cohorts of top performers. A clearer picture only emerges after combining the information gathered in all four environments.

In the PointMaze environment (Figure 6.6(a)), *Occupancy L1*, *Last State Action*, and *Custom BD* all solve the environment perfectly in all 10 trials. While *Signature Depth 3* and *20 Snapshots* each fail the task once, this is not enough for a statistically significant difference. *Random BD* and *Occupancy Sinkhorn* both perform significantly worse.

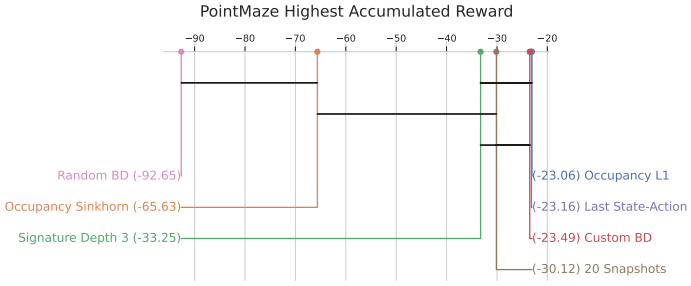
In the Walker2D environment (Figure 6.6(b)), the highest performer is *Random BD* but not significantly different from the others in the top cohort *20 Snapshots*, *Last State Action*, and *Signature Depth 3*. The two different ways to use the Occupancy measure and the *Custom BD* perform worst.

In the AntTrap environment (Figure 6.6(c)), *Random BD* is significantly worse than the others, while *Occupancy L1* and *Signature Depth 2* are significantly better than the others. All others are close and not significantly different.

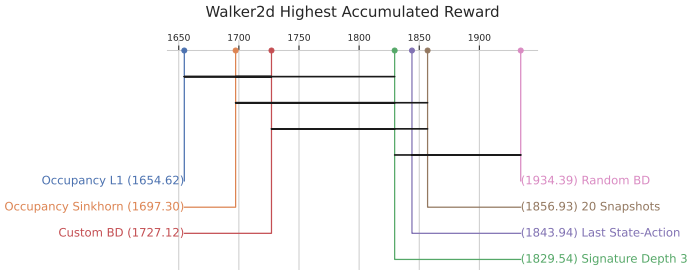
In the AntMaze environment (Figure 6.6(d)), the *Custom BD* has the highest performance but is not significantly different from *Last State Action*, *Occupancy L1*, or *20 Snapshots*.

Table 6.3: Runtime per generation in seconds based on environment and behavioural descriptor.

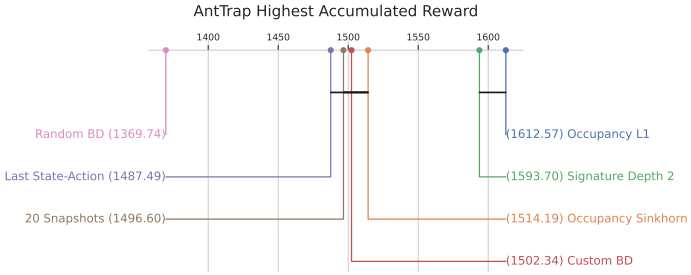
	PointMaze	Walker2D	AntTrap	AntMaze
Occupancy L1	1.65	4.59	14.82	40.24
Occupancy Sinkhorn	7.08	9.98	20.17	45.56
Signature Depth 3	1.28	15.66	-	-
Signature Depth 2	-	-	20.69	47.41
Custom BD	1.14	4.05	14.13	39.56
Last State-Action	1.17	4.07	14.26	39.67
20 Snapshots	1.21	4.33	15.75	40.98
Random BD	1.13	4.00	14.10	39.65



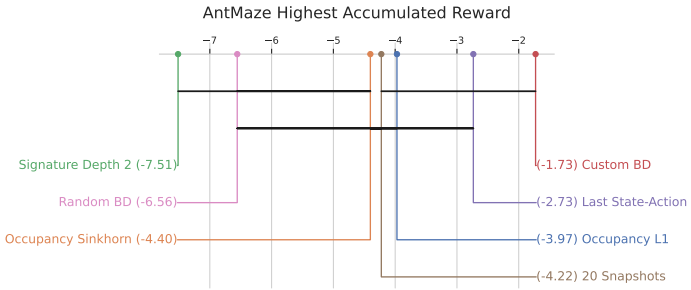
(a) PointMaze



(b) Walker2D



(c) AntTrap



(d) AntMaze

Figure 6.6: Plot of the highest performance observed in the population in different environments, each averaged over 10 runs, with different seeds to generate random numbers during the evolutionary learning process. Depicted at the last generation in a critical difference diagram, showing a horizontal bar over algorithms where a pairwise Mann–Whitney U test finds no statistically significant difference ($p < 0.05$). See also Appendix A.3.

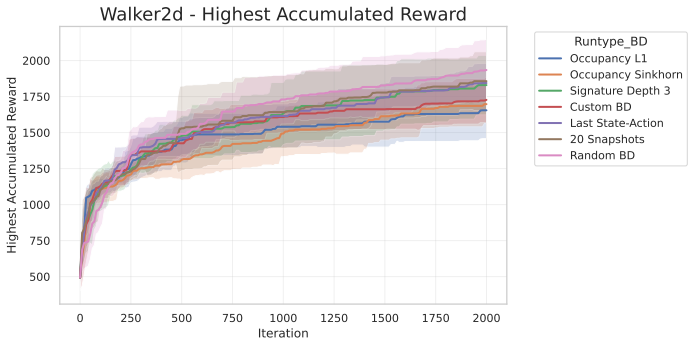
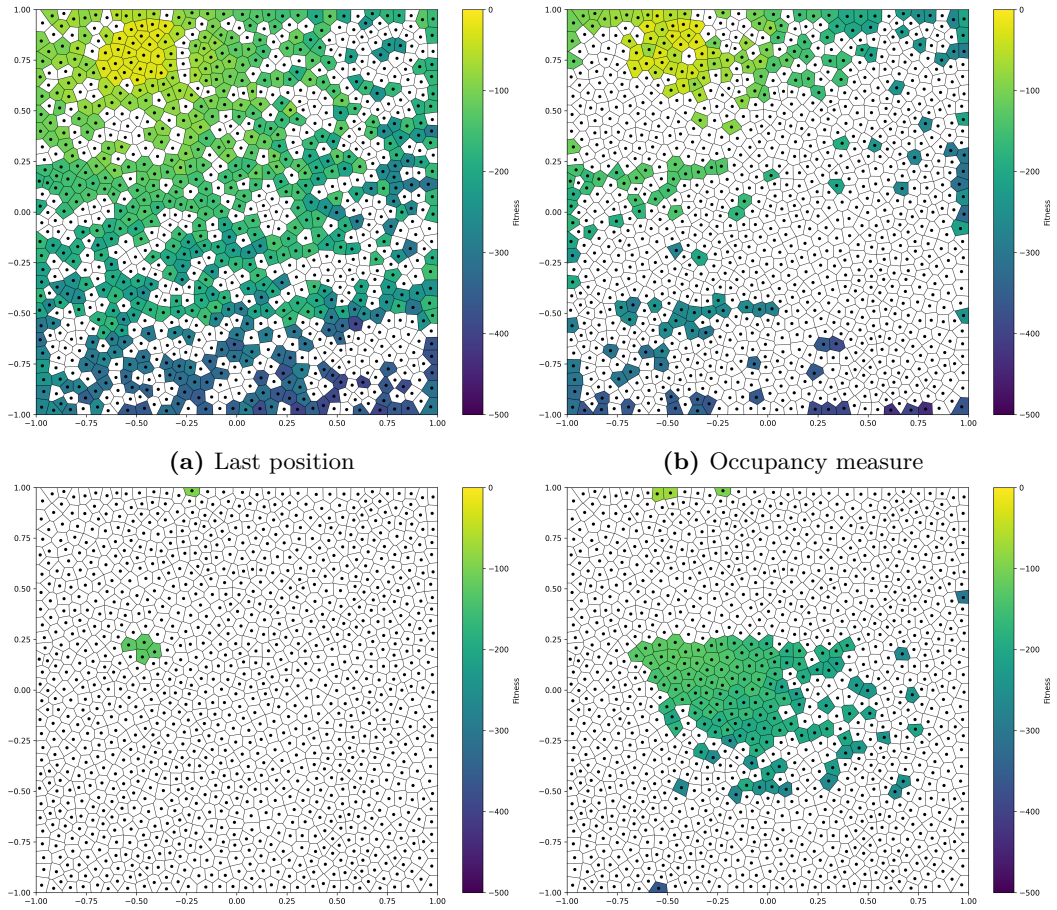


Figure 6.7: Evaluation of the best performance in the Walker2D environment, averaged over 10 runs with different initial seeds. Depicted over evolutionary generations.

Overall, in the hard exploration environments, the *Occupancy L1* is the best choice, dominating the AntTrap with *Signature Depth 2* and not being significantly different from the top choice in the other two. *Custom BD* and *Last State-Action* perform very well as expected in these maze-like configurations where the last position of the agent is critical for its success. Walker2D shows inverse results, probably because here, diverse behaviour is not as helpful. After learning to walk, it pays off only to improve on that strategy instead of cultivating different diverse strategies. The somewhat weak showing of the *Custom BD*, which emphasises different gaits, supports this interpretation. *Occupancy Sinkhorn* creates mediocre results, casting doubts on its utility for this particular application. It is unclear what aspect of that technique causes problems. Extrapolating from Figure 6.3 may suggest that the Sinkhorn distance blurs the existing differences too much. Other approaches for a fast calculation of optimal transport, such as linear optimal transport [WSB+13], could be worthwhile.

The observed behaviour may be explained based on two fundamental ideas around evolution and diversity: First, exploration and exploitation are competing objectives. A stronger emphasis on exploration may hamper the optimisation process. This is especially evident in the Walker2D environment: None of the optimisation runs in the Walker2D environment shows a converged behaviour, see Figure 6.7. All are still improving over the generations. Improvement is slowest with behavioural descriptors that explore more. The other extreme is visible in the PointMaze environment, where exploration is most important and finding optimal behaviour is simple, as soon as the environment is explored well enough. AntMaze and AntTrap fall somewhere in between the extremes: A balance between exploration and exploitation is necessary, but exploration is more important than in typical reinforcement learning problems.

In Figure 6.8, the population at the end of an optimisation run in the PointMaze environment is depicted by projecting the population into the same MAP-Elites archive based on the last state behavioural descriptor. Figure 6.8(a) visualises a population developed with the last position behavioural descriptor and shows an almost perfect coverage of the projected archive, since the original archive and the projected archive are perfectly aligned and the evolutionary algorithm has successfully developed an exhaustive population. Figure 6.8(b) visualises a population developed with the occupancy measure as the behavioural descriptor and shows a more patchy coverage that still covers the full behaviour space. The occupancy measure



(a) Last position **(b)** Occupancy measure

(c) Random BC **(d)** 20 Snapshots

Figure 6.8: Projected archives after optimisation on the PointMaze environment with the MAP-Elites algorithm while using different behavioural descriptors.

sufficiently encourages diverse behaviour that fully explores the solution space. Figure 6.8(c) visualises a population developed with the *random* behavioural descriptor, which clearly demonstrates the problem of using no meaningful behavioural descriptor in the MAP-Elites loop. The population exhibits a *crowding* phenomenon, where all members of the population have a very similar behaviour, which causes the algorithm to lose any advantage it hopes to gain from the population approach. Figure 6.8(d) visualises a population developed with the *20 snapshots* behavioural descriptor. It shows less pronounced crowding. The behavioural descriptor is somewhat useful in fostering a diverse population, but the occupancy measure seems more promising.

Second, the key to good exploration is open-ended exploration [SL15]. An effective exploration strategy is characterised by a mechanism that continues to find new ideas. This can somewhat be observed in Figure 6.9, which shows the highest performance of any individual in the population during learning in the AntTrap envirionment. After iteration 500, the top performers continue to learn, while the other two strong contenders improve almost not at all. Similarly, *Random BD* has a flat curve in all environments except the Walker2D, suggesting the diversity of the population does not suffice for continuous exploration.

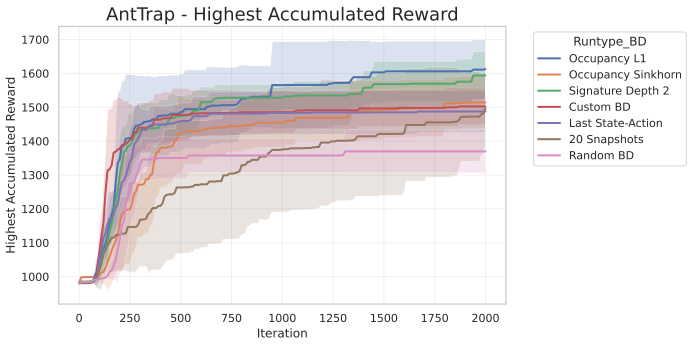
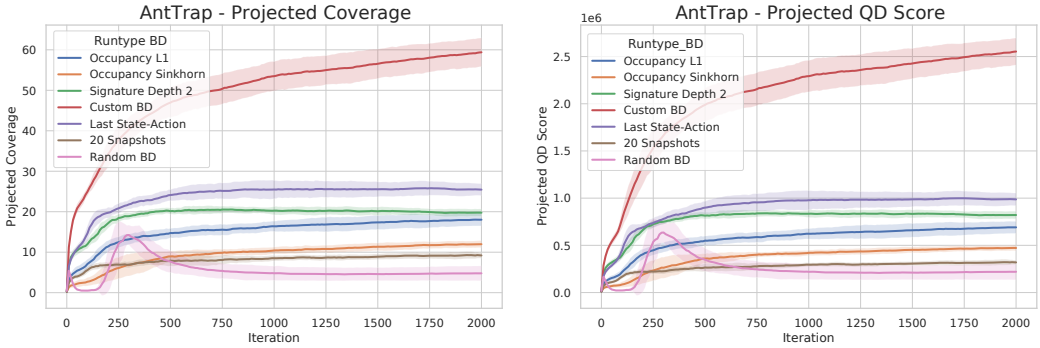


Figure 6.9: Evaluation of the highest performance of the population observed in the AntTrap environment, averaged over 10 runs with different initial seeds, depicted over evolutionary generations.



(a) Projected coverage **(b)** Projected QD-score

Figure 6.10: Evaluation of the projected coverage and projected QD-score observed in the AntTrap environment, averaged over 10 runs with different initial seeds. Depicted over evolutionary generations.

This effect can even better be observed in the projected coverage and QD-score, see Figure 6.10, for AntTrap: Clearly, *Custom BD* dominates this metric, as the projected space and the behaviour space of that metric are the same. Similarly, *Last State-Action* very quickly achieves a strong score, as the spaces are well aligned. Comparing the unsupervised methods, it can be observed that both methods based on the occupancy measure have rising scores, while the others stay flat after a short increase in the beginning. This shows how the diversity of the behaviour is still being improved over time for these two methods. Again, the inverse is true for *Random BD*, where diversity drops over time, which explains why no further improvement is made in the fitness of the best individual.

A similar effect may explain the weak showing of the signature-based behavioural descriptor in the AntMaze environment. Figure 6.11 shows the projected coverage during runs in the AntMaze environment. For the signature transform as the behavioural descriptor, it drops over time, similar to *Random BD*, if not as sharply. The projected coverage may be interpreted as a proxy for exploration in the significantly aligned custom behavioural space. If it stagnates or falls, exploration halts. However, the projected coverage rises during training for *Occupancy L1*. The root cause for the weak showing of the signature

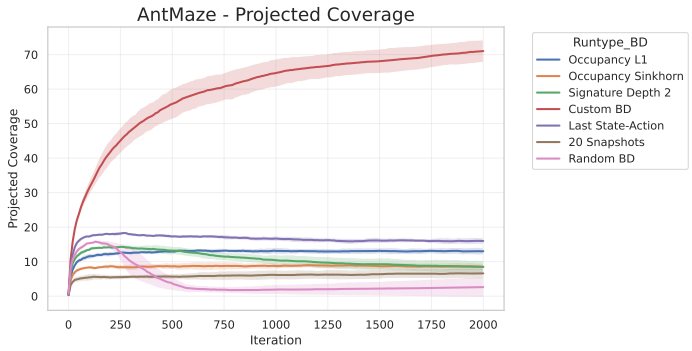


Figure 6.11: Evaluation of the projected coverage in the AntMaze environment, averaged over 10 runs with different initial seeds. Depicted over evolutionary generations.

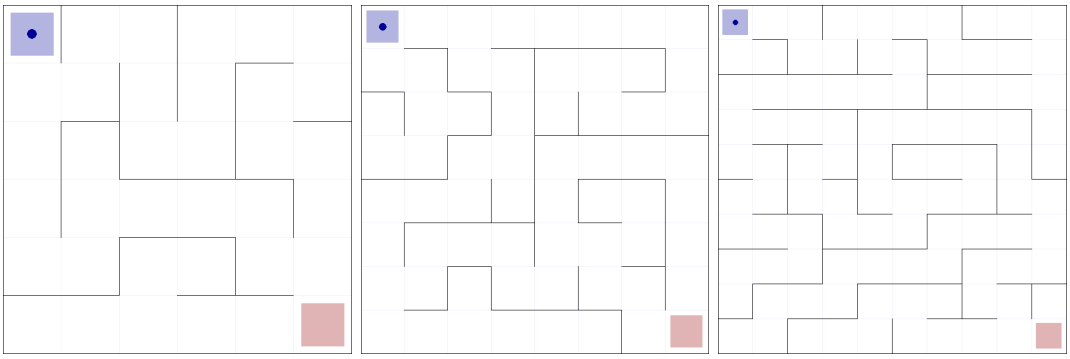
transform as a behavioural descriptor here may be that a signature of depth 2 does not sufficiently capture subtle differences in behaviours. This may actually be advantageous in the AntTrap environment, where less exploration is necessary, but a disadvantage in the AntMaze environment.

With the results at hand, it is evident that the framework itself already provides strong exploration capacities. Still, the choice of behavioural descriptor significantly impacts the results of the procedure. The results suggest that the use of the occupancy measure as a behavioural descriptor fosters diversity in a useful and flexible manner.

6.3.5 Generalised Maze Environment

From the previous study, the cleanest reflection on the exploration capabilities of a certain behavioural descriptor is given with the PointMaze environment. It is, however, a very simple maze. A more in-depth study may confirm the preliminary results. To address this, the original [FG23], procedurally generated *SquareMaze* environment offers more variety.

In this environment, just like the PointMaze, a particle is tasked to move from a starting position to a goal. The reward is defined through the distance of the particle from the goal. In each step, the change in distance is the reward. That means, moving closer to the goal gives a positive reward, and moving farther away gives a negative reward. This, of course,



(a) Size 6×6 **(b)** Size 8×8 **(c)** Size 10×10

Figure 6.12: Depiction of three SquareMaze environments of different sizes. The blue dot symbolises the position of the agent, the red square symbolises the goal.

creates deceptive reward structures. The reward is normalised, such that the maximum reward that can be achieved over a full episode is always 1.

$$r_t = \frac{\left\|s_t - s_{\text{goal}}\right\|_2 - \left\|s_{t+1} - s_{\text{goal}}\right\|_2}{\left\|s_{\text{goal}} - s_{\text{start}}\right\|_2}$$

The maze is randomly created by first parcelling the square into n by n square cells of equal size. Then, in an iterative depth-first search, walls are knocked down. Beginning with the starting cell, a random previously unvisited neighbouring cell is selected, and the wall between the two cells is knocked down. This process is restarted, beginning with the newest, already visited cell. No walls are knocked down if all neighbouring cells have been visited. This process is repeated until every cell has been visited, ensuring that a path to it has been created.

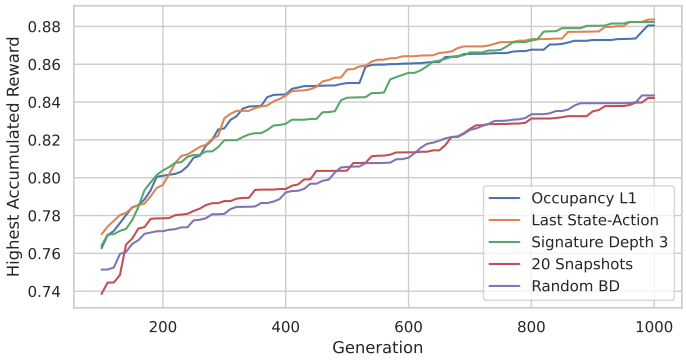
This maze is then the basis for a continuous reinforcement learning problem: The state of the environment is just the position of the particle. It always starts in the middle of the cell in the upper left corner. The action space is a vector that gives the direction and length of the next step and is capped as $(-0.5, 0.5)^2$. A full unit in movement corresponds to the length and width of one square cell. That means, traversing the length of an n by n maze would require $2n$ unobstructed movements in the direction $(0, 0.5)$.

The maze's walls obstruct the movement of the particle. To simplify the collision calculation, diagonal movement is resolved as two consecutive straight movements, where the direction with the higher absolute value is resolved first. So, the action $(0.4, 0.2)$ is resolved first as a movement of 0.4 in the vertical direction, then as a movement of 0.2 in the horizontal direction. Collision with a wall is handled by stopping the particle's movement to 0.1 before the wall.

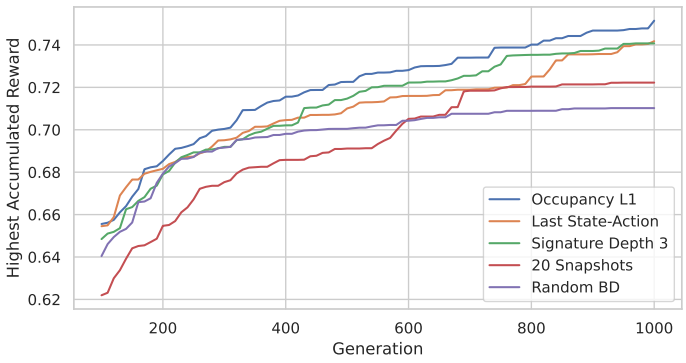
With the same settings as before and using the capability to create an arbitrary number of different mazes, the exploration capabilities gained from different behavioural can again be tested. For each of the maze sizes 6×6, 8×8, and 10×10, 100 mazes are created randomly. One of each is depicted in Figure 6.12. Then, each behavioural descriptor is tested in just one run in each of these environments. In contrast to the PointMaze, these environments are more challenging, but they also have a more gradual reward structure, as there are more than just two walls that may obstruct movement.

The learning curves in Figure 6.13 paint a very clear picture for each size and across all sizes: Both *Occupancy L1* and *Signature Depth 3* are on par with *Last State-Action*, while *20 Snapshots* is in line with the performance of *Random BD*.

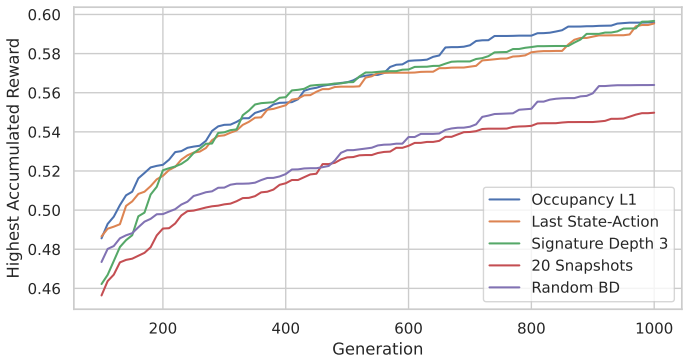
It underlines the results from the last section. The exploration provided by the unsupervised behavioural descriptors through the occupancy measure or the signature transform is at least as useful as the exploration provided by directed, custom behavioural descriptors, and in general, more efficient than the current standard approach of chaining several snapshots. The exploration capabilities of the MAP-Elites framework itself, however, are already quite strong, and the choice of behavioural descriptor plays only a secondary role.



(a) Size 6×6



(b) Size 8×8



(c) Size 10×10

Figure 6.13: Average performance of the best individual over 100 different mazes of different sizes and depending on the behavioural descriptor.

6.4 Further Applications

This chapter has laid out in detail one possible application for general behavioural descriptors in a specific MAP-Elites setup. The same principle can also work in other algorithms that require a distance measure on policies. One such example is given in Subsection 6.4.1. It can also be the basis to select a diverse subset of policies to exploit an expected advantage of a diverse set of policies, such as robustness against environmental change shown in Subsection 6.4.2. The chapter finishes with an outlook that includes a brief sketch of the incorporation of the occupancy measure with the AURORA algorithm in Subsection 6.4.3.

6.4.1 Unsupervised Dominated Novelty Search

Dominated novelty search (DNS) [BFG+25] is a recent heuristic that reconciles the novelty-first approach with the idea of quality-diversity, that is, selecting for performance in clusters of similar behaviour. The general evolutionary loop is the same as this chapter's unsupervised MAP-Elites (Section 6.2). It differs in the definition of the archive and the selection process for the next generation.

It forgoes defining a structure for the archive and, rather, at every generation picks the best N individuals out of the population of N parents and N' offspring. An alternative fitness score decides which individuals survive. It is calculated for each individual as the average distance to the k-closest "dominating" other solutions, where dominating means achieving higher performance in the original task, so the cumulative discounted reward of observed episodes. The k highest performing solutions are always selected for the next generation. A random choice is made if two solutions have the same fitness value. In some instances, this approach beats alternative archive structures with MAP-Elites.

This idea can be integrated with the generalised behavioural descriptors, for example, with the occupancy measure. To this end, the experiment from Section 6.2 was repeated with the same hyperparameters, see Appendix A.2, as applicable. The total population size is set as 200, and k, the number of dominating other solutions considered, is set as 5.

For the SquareMaze environment, see Figure 6.14, dominated novelty search outperforms the usual MAP-Elites approach slightly. The behavioural descriptor that uses the occupancy measure is just as useful for exploration as the problem-specific behavioural descriptor. However, when trying this approach on the more complex environments that combine a complicated learning task with hard exploration, the approach is clearly worse than MAP-Elites.

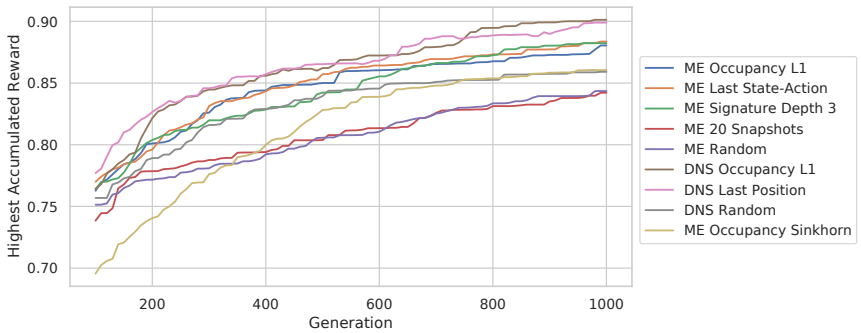


Figure 6.14: The average score for 6x6 mazes averaged over 100 instances of the SquareMaze environment.

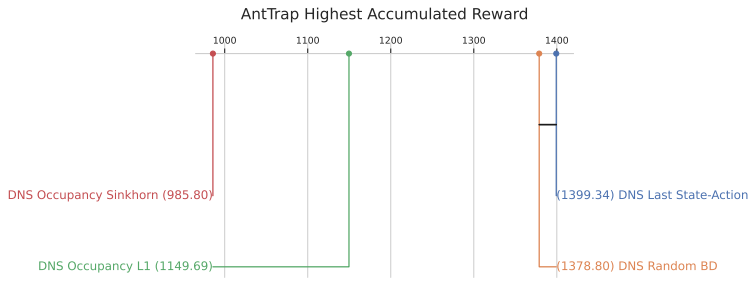


Figure 6.15: The performance of dominated novelty search for the AntTrap environment averaged over 10 runs. See also Appendix A.3.

Figure 6.15 shows the performance of dominated novelty search in the AntTrap environment. The last state-action as a behavioural descriptor is only as good as replacing it with selecting a random number. The two approaches to use the occupancy measure learn little more useful behaviour beyond standing, which incurs a total of 1000 points. Figure 6.6(c) shows the performance of MAP-Elites in the same environment, which clearly outperforms dominated novelty search.

While a definitive diagnosis of the problem is hard to achieve by only looking at behaviour in benchmarks, the problem seems to be an overemphasis on exploration at the expense of performance improvement. The learning algorithms get lost in extremely different behaviours, all of which are not viable in the sense that they may do things so differently that the robot does not even properly walk anymore. This stronger influence on exploration is useful in environments like SquareMaze, where it is not a small subspace of viable solutions that is to be explored, but rather every different new behaviour is a useful step for exploration.

Conversely, this shows the positive, stabilising influence of the structured archive that is used in MAP-Elites. The MAP-Elites loop always improves on the QD-score, and with that, a good solution that is assigned to a certain niche can only be replaced by better performance with behaviour in the same niche. An unstructured archive always has to balance exploration and exploitation in its population, which makes the algorithm sensitive to changes regarding how much exploration is healthy. Too little exploration leads to crowding (compare Figure 6.8), too little exploitation abandons the "quality" portion of the quality-diversity approach.

6.4.2 Selecting a Robust Subpopulation

A diverse population can enable a population-based learning algorithm to explore the search space better. Another expected advantage of a diverse population over a uniform population is robustness against environmental change. If there are several different solutions for the same problem, some of them may still work for a similar but changed problem setting. If there are several very similar solutions to the same problem, they probably either all fail or all succeed. This concept can be exploited to create repertoires of behaviours that can be a basis to find new solutions to a changed environment without restarting the learning process.

The validity of this idea has been demonstrated using MAP-Elites for walking gaits of a robot for a case where the change of environment and the behavioural descriptor are aligned [CCT+15]. But can a general behavioural descriptor that selects for diversity also find a more robust population?

The analysis of this chapter has shown that MAP-Elites can create diverse populations

Algorithm 7 Farthest point sampling

Require: Data X, initial subset V, target size n, metric d
  while $|V| < n$ **do**
    $V \leftarrow V \cup \{\arg\max_{x \in X} \min_{\tilde{x} \in V} d(x, \tilde{x})\}$
  return V

even with nonsensical behavioural descriptors. So, for this demonstration, instead of creating a diverse population, a diverse subpopulation is selected from larger set of policies. With each of the algorithms ARS, A2C, PPO, and TRPO, 12 policies are trained with different settings that solve the CartPole problem, totaling 48 different viable solutions. These policies are then analysed for their robustness properties (see Subsection 4.4.3). The break condition for training here was the total number of training steps instead of breaking when a solution is initially reached. That leads to different robustness properties than in Subsection 4.4.3.

With farthest point sampling [ELP+97] described in Algorithm 7, a diverse subpopulation can be sampled from this general population. Farthest point sampling is an algorithm that iteratively adds members to a subset of a given set. Again, if min or arg max are not unique, a random choice is made. Farthest point sampling selects greedily for maximum coverage of the space covered by the resulting subset. As such, it is a selection algorithm that preserves the spread of diversity in a given population. Farthest point sampling has been shown to have favourable properties selecting training data for classification and regression problems [CG24].

Figure 6.16 shows a visualisation of the population, separated according to the algorithm used to train it. It also shows which 12 individuals would be selected using farthest point sampling with the last position as the behavioural descriptor, or using the signature transform as the behavioural descriptor.

To measure the robustness of a population instead of an individual, for every change

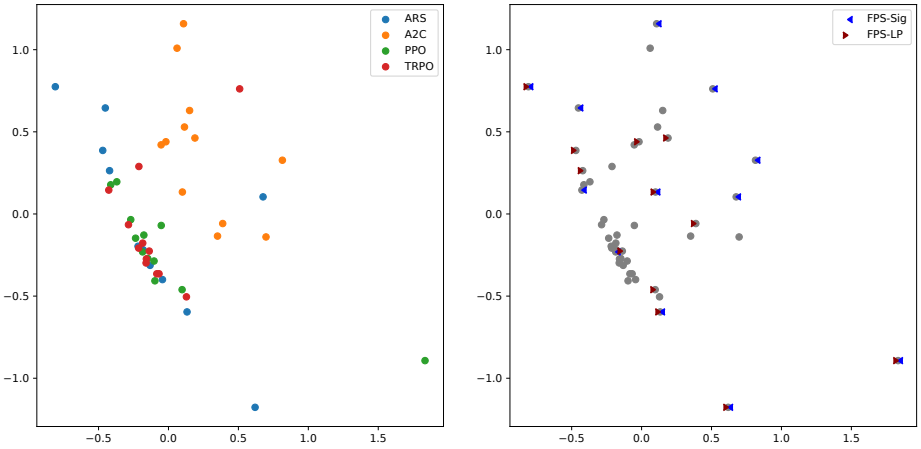


Figure 6.16: Using MDS and signature depth 3 as behavioural descriptor to visualise the full population and the choices of subpopulations by farthest point sampling either with last position or with signature as distance measure

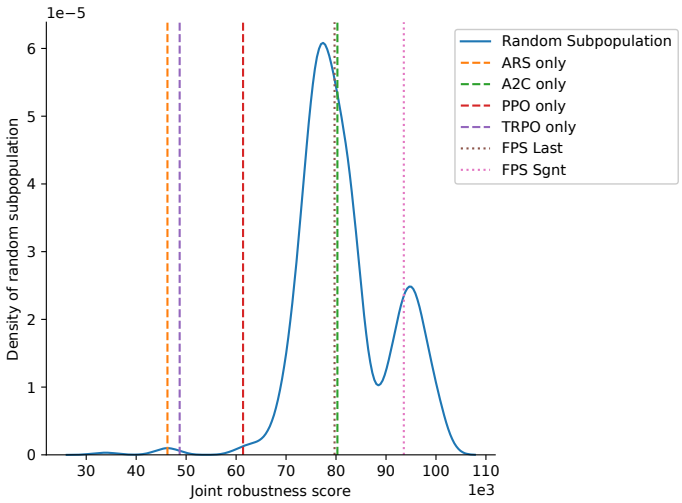


Figure 6.17: The joint robustness score of different subpopulations, indicated by vertical lines. Shown in blue as a baseline for efficient selection, the estimated distribution of the random variable that represents the process of picking 12 policies from the whole population at random and calculating the joint robustness score of that random selection.

Table 6.4: Joint robustness score of different subpopulations

Selection	JRS	Selection	JRS
ARS only	46265	FPS Sgnt	93575
A2C only	80281	FPS Last	79699
PPO only	61392	Random Mean	81602
TRPO only	48706	Full Pop.	101459

in environment, the policy that performs best in that environment counts for the *joint robustness score* (compare Definition 25)

$$\mathrm{jrs}(\Pi) = \sum_{i,j=1}^{20} \max_{\pi \in \Pi} \eta(\pi | \mathcal{M}^{\mathrm{cart}}(g_i, l_j)). \tag{6.37}$$

Taking the maximum reflects the idea of a repertoire of behaviours from which a suitable behaviour can be selected if a new environment is encountered.

Farthest point sampling with signature transform (“FPS Sgnt”) and farthest point sampling with last position (“FPS Last”) select a subpopulation of 12 policies. The joint robustness score of them is shown in Figure 6.17 and Table 6.4. As a baseline, the joint robustness score of the 12 subpopulations, with only taking policies trained with one algorithm, is also shown (“ARS only” means all 12 policies trained with ARS, and so on). In Table 6.4, the mean of selecting a random subpopulation of size and the joint robustness score of the full population are also shown. In Figure 6.17, the blue curve shows the distribution that represents the joint robustness score of picking a subpopulation of 12 policies at random that is estimated from 1000 randomly selected subpopulations.

Farthest point sampling with the last position is almost indistinguishable from picking

only A2C, but the mean of a random selection is still slightly higher. It may be correct to interpret this as a high performance of random sampling rather than a low performance of the other populations: A random selection is often hard to beat when trying to find a good representation of data. Only farthest point sampling with signature finds a subpopulation in the second bump of the blue curve in Figure 6.17. This short analysis shows that diversity in a general sense can be connected to the concept of robustness.

6.4.3 Outlook: The Value in Diversity

In the study on behavioural descriptors in MAP-Elites, it has become clear that the choice of behavioural descriptor plays a small but critical role in the algorithm's outcome. The unsupervised behavioural descriptors show a performance aquivalent to the custom behavioural descriptors that are fine-tuned to each problem, and a better performance than the standard unsupervised behavioural descriptor of using snapshots. This could improve the out-of-the-box capabilities of MAP-Elites algorithms and provide a good framework for reinforcement learning problems. While MAP-Elites is typically not as sample-efficient as standard reinforcement learning algorithms, the evolutionary loop is embarrassingly parallel, a property that can reduce wall time significantly [CMS+18]. This is in line with the bitter lesson [Sut19] that general learning algorithms that scale with increasing computing power are preferable to problem-specific solutions. With the advent of more powerful GPUs, evolutionary algorithms that parallelise well are already becoming more important, for example, a recent approach fine-tunes large language models with evolutionary strategies [QGH+25].

Most state-of-the-art applications are already very elaborate products of engineering, which makes testing new approaches a cumbersome task. The experiments in this chapter support that unsupervised quality-diversity based on a full behavioural characterisation is viable and even powerful in sponsoring exploration in reinforcement learning. The hypothesised advantages of generalised diversity, like robustness, would also merit structured analysis, which was only touched on here.

The proof of concept in this chapter utilises the MAP-Elites framework designed originally for low-dimensional behavioural descriptors. It may function even better in a framework that is more inclined to higher-dimensional spaces. For example, when working with MAP-Elites, the creation of centroids is a problematic issue in a more complicated behaviour space. An unstructured archive avoids that problem. AURORA [GC22b] takes that approach when learning behavioural descriptors through an autoencoder, and unstructured archives also avoid other problems like the necessity to bound the behaviour space a priori [BFG+25].

AURORA is also an interesting target to incorporate the occupancy measure as a selection strategy: In AURORA, an autoencoder with a low-dimensional latent space forms the basis of the behavioural descriptor. Typically, either the last state or a concatenation of snapshots forms the input of the autoencoder. The latent space representation of this selection of the observed episode is the behavioural descriptor of the policy. Clearly, such a transformation would also be possible with an occupancy-based understanding of the policies' behaviour. Of course, several technical problems would need to be solved for that. In AURORA, the autoencoder is developed alongside the learning algorithm, which means that the function serving as the behavioural descriptor changes with every iteration of the autoencoder. This makes maintaining the archive properly difficult. The canonical solution is to use an unstructured archive, which is challenging to reconcile with the approach taken here to discretise the state(-action) space.

In the context of this work, it is particularly noteworthy that the abstract behavioural distance measures introduced show a tangible, successful application. This is a strong indication that these concepts have more merit than just a theoretical foundation: They can be translated into workable algorithms that deliver competitive results. This supports the case that these techniques also work for problems that have no objective function, such as visualisation and interpretability.

CHAPTER 7

Conclusion

To view a policy through its behaviour is at the core of reinforcement learning. Because learning from experience and reinforcing behaviour ultimately means to base the learning paradigm on a behavioural understanding of the policy. This is evident in the definition of the optimisation goal, where most algorithms do not look for the optimal policy as defined by Theorem 3, but rather for an optimally performing policy under starting states as in Definition 15. It is rooted in reinforcement learning algorithms, be it through an episodic understanding as in REINFORCE (see Subsection 3.1.2) or through understanding of the policy's occupancy, which usually leads to a usage of replay buffers as in TD3 (see Subsection 3.1.5). The practical approach to set the performance of a policy as the target of optimisation stems from the necessity to limit the scope of the learning process: To learn only about states of the environment that are expected to be encountered, not to learn about any conceivable configuration.

While this idea is ingrained in reinforcement learning, it exists on a subliminal level. For example, the derivation of the policy gradient theorem speaks of finding a "locally optimal policy" [SMS+99] to mark that the proper optimality can only be reached on states visited during the learning process. The derivation of the TRPO algorithm, which uses a policy-gradient approach, identifies a policy with optimal performance simply as an "optimal policy" [SLA+15]. This sleight of hand exemplifies the conceptual identification of optimality and optimal performance, and with it the deep influence of the behavioural understanding of policies. It is therefore not a lack of alternatives or mere technical necessity that makes the behavioural understanding the main avenue to quantify distance in policies. It is rather motivated directly out of the practical and theoretical foundation of reinforcement learning.

In general, the question of how to measure distance soundly is not addressed in reinforcement learning literature. The foundational textbook by Sutton and Barto [SB18] does not pose the question at all, and contemporary research that needs some idea of distance begins the quest for a proper distance measure from scratch. For example, a contribution on multi-agent reinforcement learning [HPA+24] includes reflections about metrics similar to those in Subsection 4.4.2 and on measuring phenotype or genotype.

The present work contains theoretical reflections where the distance of policies is already conceptually found and utilised. From these reflections, concepts of behavioural distance are developed. The behaviour of a policy is understood as a random element either in the space of trajectories in the state-action space or with the occupancy measure as a random element directly in the state-action space. With this understanding, the behavioural distance between two policies is derived as an empirical distance between two random elements. To validate this approach, simple questions are posed and answered that are motivated by very basic assumptions that could not even be tested before: Are different episodes, rolled out

with the same policy, similar? Are different policies, learned through the same algorithm, similar? These questions would also merit a deeper, more structured research for which this work provides a tangible starting point.

The ability to quantify the similarity or difference between policies is clearly valuable for a better understanding of reinforcement learning and its methods. This is underlined by the robustness analysis, which reveals other, hidden properties of policies that correspond with the distances measured using the developed methods. These approaches also seem promising as a tool for analysing the learning process that goes beyond monitoring performance and loss values.

The approach to define a pairwise distance of policies through an optimal transport distance of their estimated occupancy is completely novel. This is apparent when looking at literature around quality-diversity, where distance between policies is essential, but always implemented either through complex, problem-specific, composite behavioural descriptors, or through an episodic understanding as in CVT-MAP-Elites [VCM18] or AURORA [GC22b]. This research gap is so glaring that even methods to validate whether a distance measure captures meaningful differences between policies are scant.

As an addition to the MAP-Elites algorithm, this novel point of view proves its merit in a state-of-the-art learning method. Not only does the description of a policy through the occupancy measure achieve excellent results, on par with custom metrics and surpassing other unsupervised approaches. The underlying idea is so fundamental that it would even lend to a combination with some other unsupervised approaches that reduce the dimensionality of the state space. Improving and extending the MAP-Elites algorithm, several further improvements could be tested, especially concerning the way the archive is created and structured, as the classic approaches were designed with small, simple behaviour spaces in mind.

From an analysis of populations that are diverse in a specific sense to an analysis of populations that are diverse in a general sense, another remarkable observation can be made: That there is intrinsic value in diversity as shown in exploration capabilities and in robustness of the population to environmental changes. This value manifests even if the type of diversity that is encouraged is not designed and directed towards a certain goal, but rather a natural byproduct of diverse behaviour.

As a method for visualisation, for analysing and comparing learning algorithms, and as a building block of learning algorithms, the presented ideas prove effective. The present work should be convincing that there is potential in this overlooked area of reinforcement learning, partially explored and partially just touched upon. Most of all, it should advance the goal post. Yes, it is possible to measure the pairwise distance of reinforcement learning policies in a general and meaningful way. This paves the way to both more theoretical questions, for example, about the makeup of the solution space or of the elite hypervolume, or to more practical questions, for example, about concrete mechanisms to enable an improved understanding of the results of reinforcement learning algorithms.

Bibliography

[AHT+17] Achiam, Joshua, David Held, Aviv Tamar and Pieter Abbeel: 'Constrained Policy Optimization'. *Proceedings of the 34th International Conference on Machine Learning*. ICML 2017: pp. 22–31 (cit. on pp. 3, 50).

[AHK01] Aggarwal, Charu C., Alexander Hinneburg and Daniel A. Keim: 'On the Surprising Behavior of Distance Metrics in High Dimensional Space'. *Database Theory — ICDT 2001* 2001: pp. 420–434 (cit. on pp. 55, 88).

[Alt99] Altman, Eitan: *Constrained Markov Decision Processes*. Chapman & Hall, 1999 (cit. on pp. 6, 50).

[BKS+23] Bäck, Thomas H. W., Anna V. Kononova, Bas van Stein, Hao Wang, Kirill A. Antonov, Roman T. Kalkreuth, Jacob de Nobel, Diederick Vermetten, Roy de Winter and Furong Ye: 'Evolutionary Algorithms for Parameter Optimization — Thirty Years Later'. *Evolutionary Computation* 2023, vol. 31(2): pp. 81–122 (cit. on pp. 3, 83).

[BLB+17] Bagnall, Anthony, Jason Lines, Aaron Bostrom, James Large and Eamonn Keogh: 'The Great Time Series Classification Bake off: A Review and Experimental Evaluation of Recent Algorithmic Advances'. *Data Mining and Knowledge Discovery* 2017, vol. 31(3): pp. 606–660 (cit. on p. 55).

[BFG+25] Bahlous-Boldi, Ryan, Maxence Faldor, Luca Grillotti, Hannah Janmohamed, Lisa Coiffard, Lee Spector and Antoine Cully: 'Dominated Novelty Search: Rethinking Local Competition in Quality-Diversity'. *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO 2025: pp. 104–112 (cit. on pp. 118, 122).

[BFC+20] Bard, Nolan, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare and Michael Bowling: 'The Hanabi challenge: A new frontier for AI research'. *Artificial Intelligence* 2020, vol. 280(C) (cit. on pp. 6, 25).

[BSA83] Barto, Andrew G, Richard S Sutton and Charles W Anderson: 'Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems'. *IEEE Transactions on Systems, Man, and Cybernetics* 1983, vol. 13(5): pp. 834–846 (cit. on p. 24).

[Bel57] Bellman, Richard: *Dynamic Programming*. Princeton University Press, 1957 (cit. on p. 55).

[BBC+19] Berner, Christopher et al.: *Dota 2 with Large Scale Deep Reinforcement Learning*. arXiv:1912.06680. 2019 (cit. on pp. 6, 23, 25).

[BCP+16] Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang and Wojciech Zaremba: *OpenAI Gym*. arXiv: 1606.01540. 2016 (cit. on pp. 25, 54).

[BLB+13] Buitinck, Lars, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt and Gaël Varoquaux: *API Design for Machine Learning Software: Experiences from the Scikit-Learn Project*. arXiv:1309.0238. 2013 (cit. on p. 143).

[Cha10] Chalmers, David J.: 'The Singularity: A Philosophical Analysis'. *Journal of Consciousness Studies* 2010, vol. 17(9-10): pp. 7–65 (cit. on p. 1).

[CLB+24] Chalumeau, Félix, Bryan Lim, Raphael Boige, Maxime Allard, Luca Grillotti, Manon Flageat, Valentin Macé, Guillaume Richard, Arthur Flajolet, Thomas Pierrot et al.: 'QDax: A Library for Quality-Diversity and Population-Based Algorithms with Hardware Acceleration'. *Journal of Machine Learning Research* 2024, vol. 25(108): pp. 1–16 (cit. on pp. 88, 96, 106, 107).

[CK26] Chevyrev, Ilya and Andrey Kormilitzin: 'A Primer on the Signature Method in Machine Learning'. *Signature Methods in Finance: An Introduction with Computational Applications*. Springer, 2026: pp. 3–64 (cit. on p. 58).

[CG24] Climaco, Paolo and Jochen Garcke: 'On Minimizing the Training Set Fill Distance in Machine Learning Regression'. *Journal of Data-Centric Machine Learning Research* 2024, vol. 1 (cit. on p. 120).

[CTC25] Coiffard, Lisa, Paul Templier and Antoine Cully: 'Overcoming Deceptiveness in Fitness Optimization with Unsupervised Quality-Diversity'. *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO 2025: pp. 122–130 (cit. on pp. 51, 96).

[CMS+18] Conti, Edoardo, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley and Jeff Clune: 'Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents'. *Advances in Neural Information Processing Systems*. NeuRIPS 2018, vol. 31: pp. 5032–5043 (cit. on pp. 51, 85, 86, 94, 122).

[CB16] Coumans, Erwin and Yunfei Bai: *PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning*. 2016. URL: http://pybullet.org/, visited on 01/05/2026 (cit. on pp. 25, 27).

[CCT+15] Cully, Antoine, Jeff Clune, Danesh Tarapore and Jean-Baptiste Mouret: 'Robots That Can Adapt like Animals'. *Nature* 2015, vol. 521: pp. 503–507 (cit. on pp. 1, 3, 51, 53, 96, 119).

[CD18] Cully, Antoine and Yiannis Demiris: 'Quality and Diversity Optimization: A Unifying Modular Framework'. *IEEE Transactions on Evolutionary Computation* 2018, vol. 22(2): pp. 245–259 (cit. on pp. 51, 91).

[Cut13] Cuturi, Marco: 'Sinkhorn Distances: Lightspeed Computation of Optimal Transport'. *Advances in Neural Information Processing Systems*. NIPS 2013, vol. 26(2): pp. 2292–2300 (cit. on p. 99).

[DDS+09] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei: 'ImageNet: A Large-Scale Hierarchical Image Database'. *2009 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR 2009: pp. 248–255 (cit. on pp. 1, 7, 24).

[EHL+20] Ecoffet, Adrien, Joost Huizinga, Joel Lehman, Kenneth O. Stanley and Jeff Clune: 'First Return, Then Explore'. *Nature* 2020, vol. 590: pp. 580–586 (cit. on pp. 6, 89).

[ELP+97] Eldar, Yuval, Michael Lindenbaum, Moshe Porat and Yehoshua Zeevi: 'The Farthest Point Strategy for Progressive Image Sampling'. *IEEE Transactions on Image Processing* 1997, vol. 6(9): pp. 1305–1315 (cit. on p. 120).

[FG23] Feiden, Arno and Jochen Garcke: 'Overcoming Deceptive Rewards with Quality-Diversity'. *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. GECCO Companion 2023: pp. 279–282. doi: 10.1145/3583133.3590741 (cit. on pp. 4, 95, 115).

[FG25] Feiden, Arno and Jochen Garcke: 'Diversity in Reinforcement Learning Through the Occupancy Measure'. *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO 2025: pp. 1235–1245. doi: 10.1145/3712256.3726337 (cit. on pp. 95, 97, 106).

[FPG24] Feiden, Arno, Biagio Paparella and Jochen Garcke: 'Generalizing Diversity with the Signature Transform'. *Proceedings of the Genetic and Evolutionary Computation Conference Companion* 2024: pp. 275–278. doi: 10.1145/3638530.3654295 (cit. on pp. 4, 76, 97).

[Fei11] Feinberg, Eugene A.: 'Total Expected Discounted Reward MDPs: Existence of Optimal Policies'. *Wiley Encyclopedia of Operations Research and Management Science*. Wiley, 2011 (cit. on p. 16).

[FCG+21] Flamary, Rémi et al.: 'POT: Python Optimal Transport'. *Journal of Machine Learning Research* 2021, vol. 22(78): pp. 1–8 (cit. on p. 59).

[FN21] Fontaine, Matthew and Stefanos Nikolaidis: 'Differentiable Quality Diversity'. *Advances in Neural Information Processing Systems 34*. NeuRIPS 2021, vol. 34: pp. 10040–10052 (cit. on p. 94).

[FTN+20] Fontaine, Matthew C., Julian Togelius, Stefanos Nikolaidis and Amy K. Hoover: 'Covariance Matrix Adaptation for the Rapid Illumination of Behavior Space'. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO 2020: pp. 94–102 (cit. on pp. 51, 90).

[FFR+21] Freeman, C. Daniel, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch and Olivier Bachem: *Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation*. arXiv:2106.13281. 2021 (cit. on pp. 25, 27, 73, 106).

[FHM18] Fujimoto, Scott, Herke van Hoof and David Meger: 'Addressing Function Approximation Error in Actor-Critic Methods'. *Proceedings of the 35th International Conference on Machine Learning*. ICML 2018: pp. 1582–1591 (cit. on pp. 21, 43, 90).

[GC22a] Grillotti, Luca and Antoine Cully: 'Relevance-Guided Unsupervised Discovery of Abilities with Quality-Diversity Algorithms'. *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO 2022: pp. 77–85 (cit. on pp. 51, 53, 96).

[GC22b] Grillotti, Luca and Antoine Cully: 'Unsupervised Behavior Discovery with Quality-Diversity Optimization'. *IEEE Transactions on Evolutionary Computation* 2022, vol. 26(6): pp. 1539–1552 (cit. on pp. 51, 53–55, 88, 96, 122, 126).

[GYZ+25] Guo, Daya et al.: 'DeepSeek-R1 Incentivizes Reasoning in LLMs through Reinforcement Learning'. *Nature* 2025, vol. 645(8081): pp. 633–638 (cit. on pp. 1, 6).

[HZA+18] Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel and Sergey Levine: 'Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor'. *Proceedings of the 35th International Conference on Machine Learning*. ICML 2018: pp. 1861–1870 (cit. on pp. 6, 27, 44, 54, 73).

[HO01] Hansen, Nikolaus and Andreas Ostermeier: 'Completely Derandomized Self-Adaptation in Evolution Strategies'. *Evolutionary Computation* 2001, vol. 9(2): pp. 159–195 (cit. on pp. 84, 90).

[HDS+18] Hasselt, Hado van, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat and Joseph Modayil: *Deep Reinforcement Learning and the Deadly Triad*. arxiv:1812.02648. 2018 (cit. on p. 17).

[Hel21] Helfenstein, Felix: 'Benchmarking Deep Reinforcement Learning Algorithms'. Bachelor thesis. Technische Universität Darmstadt, 2021 (cit. on p. 28).

[HIB+18] Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup and David Meger: 'Deep Reinforcement Learning that Matters'. *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI 2018, vol. 32(1): pp. 3207–3214 (cit. on p. 28).

[HL01] Hillier, Frederick S. and Gerald J. Lieberman: *Introduction to Operations Research*. McGraw-Hill, 2001 (cit. on p. 59).

[HGA+21] Holzleitner, Markus, Lukas Gruber, José Arjona-Medina, Johannes Brandstetter and Sepp Hochreiter: 'Convergence Proof for Actor-Critic Methods Applied to PPO and RUDDER'. *Transactions on Large-Scale Data- and Knowledge-Centered Systems* 2021, vol. 48: pp. 105–130 (cit. on pp. 6, 18, 24).

[HPA+24] Hu, Tianyi, Zhiqiang Pu, Xiaolin Ai, Tenghai Qiu and Jianqiang Yi: 'Measuring Policy Distance for Multi-Agent Reinforcement Learning'. *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*. AAMAS 2024: pp. 834–842 (cit. on pp. 6, 125).

[IML+24] Iraki, Tarek, Lukas Morand, Norbert Link, Stefan Sandfeld and Dirk Helm: 'Accurate Distances Measures and Machine Learning of the Texture-Property Relation for Crystallographic Textures Represented by One-Point Statistics'. *Modelling and Simulation in Materials Science and Engineering* 2024, vol. 32(5) (cit. on p. 99).

[KL02] Kakade, Sham M. and John Langford: 'Approximately Optimal Approximate Reinforcement Learning'. *Proceedings of the Nineteenth International Conference on Machine Learning*. ICML 2002: pp. 267–274 (cit. on p. 39).

[Kru64] Kruskal, J. B.: 'Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis'. *Psychometrika* 1964, vol. 29(2): pp. 115–129 (cit. on p. 143).

[LRD23] Laidlaw, Cassidy, Stuart Russell and Anca Dragan: 'Bridging Reinforcement Learning Theory and Practice with the Effective Horizon'. *Advances in Neural Information Processing Systems*. NeurIPS 2023, vol. 33: pp. 58953–59007 (cit. on p. 26).

[LS20] Lattimore, Tor and Csaba Szepesvári: *Bandit Algorithms*. Cambridge University Press, 2020 (cit. on pp. 1, 7, 13).

[LBB+98] LeCun, Yann, Léon Bottou, Yoshua Bengio and Patrick Haffner: 'Gradient-Based Learning Applied to Document Recognition'. *Proceedings of the IEEE* 1998, vol. 86(11): pp. 2278–2324 (cit. on p. 24).

[Lee77] Leeuw, Jan de: 'Application of Convex Analysis to Multidimensional Scaling'. *Recent Developments in Statistics*. Proceedings of the European Meeting of Statisticians 1977: pp. 133–145 (cit. on p. 143).

[LS08] Lehman, Joel and Kenneth Stanley: 'Exploiting Open-Endedness to Solve Problems through the Search for Novelty'. *Artificial Life*. ALIFE 2008, vol. 11: pp. 329–336 (cit. on pp. 2, 3, 51, 85).

[LS11a] Lehman, Joel and Kenneth Stanley: 'Evolving a Diversity of Creatures through Novelty Search and Local Competition'. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO 2011: pp. 211–218 (cit. on pp. 86, 95).

[LS11b] Lehman, Joel and Kenneth O. Stanley: 'Abandoning Objectives: Evolution through the Search for Novelty Alone'. *Evolutionary Computation* 2011, vol. 19(2): pp. 189–223 (cit. on pp. 51, 92, 93, 107).

[LVS12] Lewis, Frank L., Draguna L. Vrabie and Vassilis L. Syrmos: *Optimal Control*. Wiley, 2012 (cit. on p. 8).

[LNS+24] Liao, Shujian, Hao Ni, Marc Sabate-Vidales, Lukasz Szpruch, Magnus Wiese and Baoren Xiao: 'Sig-Wasserstein GANs for Conditional Time Series Generation'. *Mathematical Finance* 2024, vol. 34(2): pp. 622–670 (cit. on p. 58).

[Lin92] Lin, Long-Ji: 'Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching'. *Machine Learning* 1992, vol. 8(3–4): pp. 293–321 (cit. on p. 19).

[MGR18] Mania, Horia, Aurelia Guy and Benjamin Recht: 'Simple Random Search of Static Linear Policies Is Competitive for Reinforcement Learning'. *Advances in Neural Information Processing Systems*. NeuRIPS 2018, vol. 31: pp. 1805–1814 (cit. on pp. 26, 27).

[MCD+25] McLaughlin Robertson, Samuel, Thang D. Chu, Bo Dai, Dale Schuurmans, Csaba Szepesvári and Jincheng Mei: 'REINFORCE Converges to Optimal Policies with Any Learning Rate'. *Advances in Neural Information Processing Systems*. NeurIPS 2025, vol. 35 (cit. on pp. 6, 36).

[MBM+16] Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver and Koray Kavukcuoglu: 'Asynchronous Methods for Deep Reinforcement Learning'. *Proceedings of The 33rd International Conference on Machine Learning*. ICML 2016: pp. 1928–1937 (cit. on p. 45).

[MKS+15] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski et al.: 'Human-Level Control through Deep Reinforcement Learning'. *Nature* 2015, vol. 518(7540): pp. 529–533 (cit. on pp. 5, 6, 12, 19).

[Moo65] Moore, Gordon E.: 'Cramming More Components onto Integrated Circuits'. *Electronics* 1965, vol. 38(8): pp. 114–117 (cit. on pp. 1, 109).

[MFK+20] Morrill, James, Adeline Fermanian, Patrick Kidger and Terry Lyons: *A Generalised Signature Method for Multivariate Time Series Feature Extraction*. arXiv:2006.00873. 2020 (cit. on p. 59).

[Mou15] Mouret, Jean-Baptiste: *A Behavior-Performance Map Containing Many Different Types of Walking Gaits*. 2015. URL: https://www.youtube.com/watch?v=IHQgnpSphEI, visited on 01/05/2026 (cit. on p. 53).

[MC15] Mouret, Jean-Baptiste and Jeff Clune: *Illuminating Search Spaces by Mapping Elites*. arXiv:1504.04909. 2015 (cit. on pp. 2, 3, 51, 54, 55, 87, 93, 95, 101).

[NC21] Nilsson, Olle and Antoine Cully: 'Policy Gradient Assisted MAP-Elites'. *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO 2021: pp. 866–875 (cit. on pp. 90, 101, 105).

[Pao21] Paolo, Giuseppe: 'Learning in Sparse Rewards Setting through Quality Diversity Algorithms'. PhD thesis. Sorbonne Université, 2021 (cit. on pp. 51, 53, 96, 97).

[PLC+20] Paolo, Giuseppe, Alban Laflaquière, Alexandre Coninx and Stéphane Doncieux: 'Unsupervised Learning and Exploration of Reachable Outcome Space'. *2020 IEEE International Conference on Robotics and Automation*. ICRA 2020: pp. 2379–2385 (cit. on pp. 53–55, 96).

[PRB+22] Pierrot, Thomas, Guillaume Richard, Karim Beguir and Antoine Cully: 'Multi-Objective Quality Diversity Optimization'. *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO 2022: pp. 139–147 (cit. on p. 91).

[PSS16a] Pugh, Justin K., L. B. Soros and Kenneth O. Stanley: 'Searching for Quality Diversity When Diversity is Unaligned with Quality'. *Parallel Problem Solving from Nature*. PPSN 2016, vol. 14: pp. 880–889 (cit. on p. 95).

[PSS+15] Pugh, Justin K., L. B. Soros, Paul A. Szerlip and Kenneth O. Stanley: 'Confronting the Challenge of Quality Diversity'. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO 2015: pp. 967–974 (cit. on pp. 51, 86, 90, 91).

[PSS16b] Pugh, Justin K., Lisa B. Soros and Kenneth O. Stanley: 'Quality Diversity: A New Frontier for Evolutionary Computation'. *Frontiers in Robotics and AI* 2016, vol. 3 (cit. on pp. 2, 3, 51, 55, 107).

[Put94] Puterman, Martin L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994 (cit. on pp. 8, 49).

[QGH+25] Qiu, Xin, Yulu Gan, Conor F Hayes, Qiyao Liang, Elliot Meyerson, Babak Hodjat and Risto Miikkulainen: *Evolution Strategies at Scale: LLM Fine-Tuning Beyond Reinforcement Learning*. arXiv:2509.24372. 2025 (cit. on pp. 1, 6, 122).

[Rec73] Rechenberg, Ingo: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973 (cit. on p. 84).

[RSP+22] Replogle, Joseph M. et al.: 'Mapping information-rich genotype-phenotype landscapes with genome-scale Perturb-seq'. *Cell* 2022, vol. 185(14): pp. 2559–2575 (cit. on p. 31).

[RRM23] Roser, Max, Hannah Ritchie and Edouard Mathieu: *What is Moore's Law?* 2023. URL: https://archive.ourworldindata.org/20260202-130411/moores-law.html, visited on 13/02/2026 (cit. on p. 1).

[RGB11] Ross, Stéphane, Geoffrey Gordon and Drew Bagnell: 'A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning'. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. AISTATS 2011: pp. 627–635 (cit. on p. 47).

[RFL+21] Ruiz, Alejandro Pasos, Michael Flynn, James Large, Matthew Middlehurst and Anthony Bagnall: 'The Great Multivariate Time Series Classification Bake off: A Review and Experimental Evaluation of Recent Algorithmic Advances'. *Data Mining and Knowledge Discovery* 2021, vol. 35(2): pp. 401–449 (cit. on pp. 7, 55, 56).

[SC07] Salvador, Stan and Philip Chan: 'Toward Accurate Dynamic Time Warping in Linear Time and Space'. *Intelligent Data Analysis* 2007, vol. 11(5): pp. 561–580 (cit. on p. 56).

[SH23] Scarton, Ludovico and Alexander Hagg: 'On the Suitability of Representations for Quality Diversity Optimization of Shapes'. *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO 2023: pp. 963–971 (cit. on p. 87).

[SLA+15] Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan and Philipp Moritz: 'Trust Region Policy Optimization'. *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. ICML 2015: pp. 1889–1897 (cit. on pp. 3, 19, 23, 47, 125).

[SML+16] Schulman, John, Philipp Moritz, Sergey Levine, Michael I. Jordan and Pieter Abbeel: 'High-Dimensional Continuous Control Using Generalized Advantage Estimation'. *4th International Conference on Learning Representations*. ICLR 2016 (cit. on pp. 25, 27).

[SWD+17] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford and Oleg Klimov: *Proximal Policy Optimization Algorithms*. arXiv:1707.06347. 2017 (cit. on pp. 2, 6, 24, 27, 49, 54).

[SBD+08] Secretan, Jimmy, Nicholas Beato, David B. D Ambrosio, Adelein Rodriguez, Adam Campbell and Kenneth O. Stanley: 'Picbreeder: Evolving Pictures Collaboratively Online'. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI 2008: pp. 1759–1768 (cit. on p. 92).

[SHS+17] Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan and Demis Hassabis: *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. arXiv:1712.01815. 2017 (cit. on pp. 1, 6, 7, 23).

[SLH+14] Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra and Martin Riedmiller: 'Deterministic Policy Gradient Algorithms'. *Proceedings of the 31st International Conference on International Conference on Machine Learning*. ICML 2014: pp. I–387–I–395 (cit. on p. 43).

[ŞB04] Şimşek, Özgür and Andrew G. Barto: 'Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning'. *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML 2004: p. 95 (cit. on p. 93).

[SL15] Stanley, Kenneth O. and Joel Lehman: *Why Greatness Cannot Be Planned: The Myth of the Objective*. Springer, 2015 (cit. on pp. 92, 113).

[SM02] Stanley, Kenneth O. and R. Miikkulainen: 'Evolving Neural Networks through Augmenting Topologies'. *Evolutionary Computation* 2002, vol. 10(2): pp. 99–127 (cit. on pp. 18, 85).

[Sut19] Sutton, Rich: *The Bitter Lesson*. 2019. url: http://www.incompleteideas.net/IncIdeas/BitterLesson.html, visited on 01/05/2026 (cit. on pp. 1, 7, 109, 122).

[Sut25] Sutton, Rich: *The Oak Architecture: A Vision of SuperIntelligence from Experience*. 2025. url: https://www.youtube.com/watch?v=gEbbGyNkR2U, visited on 13/02/2026 (cit. on p. 6).

[SB18] Sutton, Richard S. and Andrew G. Barto: *Reinforcement Learning: An Introduction*. MIT Press, 2018 (cit. on pp. 2, 5, 6, 12, 17, 19, 21, 48, 54, 93, 125).

[SMS+99] Sutton, Richard S., David McAllester, Satinder Singh and Yishay Mansour: 'Policy Gradient Methods for Reinforcement Learning with Function Approximation'. *Advances in Neural Information Processing Systems*. NIPS 1999, vol. 12: pp. 1057–1063 (cit. on pp. 33, 40, 125).

[SBS08] Syed, Umar, Michael Bowling and Robert E. Schapire: 'Apprenticeship Learning Using Linear Programming'. *Proceedings of the 25th International Conference on Machine Learning*. ICML 2008: pp. 1032–1039 (cit. on pp. 37, 38).

[Sze23] Szepesvári, Csaba: *Planning in MDPs*. 2023. url: https://rltheory.github.io/lecture-notes/planning-in-mdps, visited on 01/05/2026 (cit. on pp. 6, 13).

[Tak81] Takens, Floris: 'Detecting strange attractors in turbulence'. *Dynamical Systems and Turbulence* 1981: pp. 366–381 (cit. on p. 20).

[Ter19] Terpilowski, Maksim A.: 'Scikit-Posthocs: Pairwise Multiple Comparison Tests in Python'. *Journal of Open Source Software* 2019, vol. 4(36): p. 1169 (cit. on p. 145).

[Tho11] Thorndike, Edward L.: *Animal Intelligence: Experimental Studies*. Macmillan, 1911 (cit. on p. 5).

[TKT+24] Towers, Mark et al.: *Gymnasium: A Standard Interface for Reinforcement Learning Environments*. arXiv:2407.17032. 2024 (cit. on pp. 25, 27).

[VM18] Vassiliades, Vassiiis and Jean-Baptiste Mouret: 'Discovering the Elite Hypervolume by Leveraging Interspecies Correlation'. *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO 2018: pp. 149–156 (cit. on pp. 87, 90).

[VCM18] Vassiliades, Vassilis, Konstantinos Chatzilygeroudis and Jean-Baptiste Mouret: 'Using Centroidal Voronoi Tessellations to Scale Up the Multidimensional Archive of Phenotypic Elites Algorithm'. *IEEE Transactions on Evolutionary Computation* 2018, vol. 22(4): pp. 623–630 (cit. on pp. 51, 54, 87, 95, 96, 101, 126).

[Vil09] Villani, Cédric: *Optimal Transport: Old and New*. Springer, 2009 (cit. on p. 59).

[VBC+19] Vinyals, Oriol et al.: 'Grandmaster level in StarCraft II using multi-agent reinforcement learning'. *Nature* 2019, vol. 575(7782): pp. 350–354 (cit. on pp. 6, 23).

[WCA+19] Wang, Ruohan, Carlo Ciliberto, Pierluigi Vito Amadori and Yiannis Demiris: 'Random Expert Distillation: Imitation Learning via Expert Policy Support Estimation'. *Proceedings of the 36th International Conference on Machine Learning*. ICML 2019: pp. 6536–6544 (cit. on pp. 3, 47).

[WSB+13] Wang, Wei, Dejan Slepčev, Saurav Basu, John A. Ozolek and Gustavo K. Rohde: 'A Linear Optimal Transportation Framework for Quantifying and Visualizing Variations in Sets of Images'. *International Journal of Computer Vision* 2013, vol. 101(2): pp. 254–269 (cit. on p. 112).

[Was21] Waskom, Michael L.: 'Seaborn: Statistical Data Visualization'. *Journal of Open Source Software* 2021, vol. 6(60): p. 3021 (cit. on p. 145).

[Wat89] Watkins, Christopher: 'Learning From Delayed Rewards'. PhD theses. Royal Holloway University of London, 1989 (cit. on p. 19).

[Wil92] Williams, Ronald J.: 'Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning'. *Machine Learning* 1992, vol. 8(3–4): pp. 229–256 (cit. on p. 35).

[ZKK+24] Zare, Maryam, Parham M. Kebria, Abbas Khosravi and Saeid Nahavandi: 'A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges'. *IEEE Transactions on Cybernetics* 2024, vol. 54(12): pp. 7173–7186 (cit. on p. 46).

List of Figures

List of Tables

List of Algorithms

A Supplementary Material

A.1 Multidimensional Scaling

In this work, low-dimensional visualisations are implemented with metric multidimensional scaling (MDS). This technique represents the original distances as well as possible, which makes it a visualisation that supports intuitive understanding, which is good for an initial exploration of different distance measurements. Given a distance matrix $D = \{d_{ij}\}_{i,j=1,\dots,N}$ for N points, the embedded points $\{z_i\}_{i=1,\dots,N}$ minimise the *raw stress* function that is defined as

$$\sum_{0<i<j<=N} \left(\|z_i - z_j\|_2 - d_{ij} \right)^2 \tag{A.1}$$

In practice, these points are found with an iterative procedure, the *SMACOF algorithm* (Scaling by MAjorizing a COmplicated Function) [Lee77] as implemented in scikit-learn [BLB+13]. This algorithm, if successful, will find the embedding that minimises the stress. To judge how well an embedding captures the original distance matrix, the minimised stress is normalised by dividing the raw stress in Equation (A.1) by $\sum_{0<i<j<=N} d_{ij}^2$ which gives the *normalised stress*, a value in [0,1]. A classic heuristic [Kru64] categorises the goodness of the fit, described in Table A.1.

Table A.1: Relationship between normalised stress and quality of fit in a classic heuristic [Kru64]

Stress (%)	**Goodness of Fit**
20%	Poor
10%	Fair
5%	Good
2½%	Excellent
0%	”Perfect”

A.2 Hyperparameters for Unsupervised PGA-ME

All hyperparameters for the experiments in Chapter 6 are summarised in Table A.2. They are subdivided by the *algorithmic section* column. *Initial CVT samples* is the number of uniform samples taken in the behaviour space Φ to create the *distinct centroids for archive* centroids for the tessellation of behaviour space, see Subsection 5.2.1. Because the calculation is too complex, for the Sinkhorn distance, this number is reduced, see Subsection 6.1.4. The state-action space is discretised to calculate an estimate of the occupancy measure of a policy again with a centroidal Voronoi tessellation. The number of centroids and with that the number of bins is *distinct centroids for occupancy*. *Population size* is the number of policies that form the population, and *new generation size* is the number of offspring created during each new generation, see Section 5.2. *ratio of PG updates to IsoLine* refers to mixing of two ways to create offspring, either using the IsoLine or policy gradient (PG) updates, see Subsection 5.2.3, where the *isoLine updates* and *policy gradient, ME* categories are explained too. For an explanation of the hyperparameters in category *policy gradient, TD3*, see Subsection 3.1.5.

The neural networks that constitute the architecture of the policies and the critics have two hidden layers, both of size 16 for the PointMaze and SquareMaze environment, and size 64 for all the others.

Table A.2: Hyperparameters for the experiments on PGA-ME in Chapter 6. For the values in *archive definition*, see Subsection 5.2.1. For *evolutionary loop*, see Section 5.2. For *Policy gradient, ME*, see 5.2.3. For *policy gradient, TD3*, see Subsection 3.1.5.

Algorithmic section	**Parameter**	**Symbol**	**Value**
Archive definition	Initial CVT samples	$n_{\text{cvt}}^{\text{sample}}$	50,000
	Initial CVT samples, Sinkhorn	$n_{\text{cvt}}^{\text{sample}}$	10,000
	Distinct centroids for occupancy	M^{occ}	500
	Distinct centroids for archive	M	1,000
Evolutionary loop	New generation size	N_{pop}	128
	Ratio of PG updates to IsoLine	-	1:1
	Total generations	-	2000
IsoLine updates	Iso σ	σ_{iso}	0.05
	Line σ	σ_{line}	0.1
Policy gradient, ME	Greedy learning rate	α_{greedy}	0.0003
	Critic training steps	N_{critic}	300
	Policy training steps	N_{policy}	100
Policy gradient, TD3	Discount factor	γ	0.99
	Replay buffer size	N_{buff}	1,000,000
	Critic learning rate for PG	α_{critic}	0.0003
	Policy learning rate	α_{policy}	0.001
	Policy noise standard deviation	σ	0.2
	Noise clip value	c	0.5
	Transitions batch size	N_{batch}	256
	Soft τ update	τ	0.005
	Policy update delay	n_{delay}	2

A.3 Plots and Statistical Significance

Two types of plot may require some clarification. First, line plots as in Figure 2.6 are empirical values over several time series. The bold line in the middle reports the average value of these time series. In this work, the transparent area of the same colour around the bold line always reports the 95% bootstrap confidence intervals of the data [Was21].

Second, critical difference diagrams report mean values on a one-dimensional scale and statistical significance. In Figure 6.6, that is the mean performance of the best policy found over several seeds for random number generation during training. Statistical significance is calculated according to a pairwise Mann–Whitney U test with $p < 0.05$. A horizontal bar connecting the values indicates a group of statistically insignificant results [Ter19].

The Mann-Whitney U test is a nonparametric statistical test. It relates two populations X_1, X_2, where each member of the population has some target value assigned in $\mathbb{R}$. The null hypothesis is that the data of that target value in X_1 and X_2 is sampled from the same distribution. This target value is, for example, the performance of the output of a learning algorithm. A rejected null hypothesis suggests that the data follows different distributions, e.g., that a change in the workings of the algorithm caused a change in the outcome of the algorithm.

The test first ranks the members of both populations according to the target value. If there is a tie, all members that are tied share a rank that is the mean of the ranks they are tied for. For example, if X_1 has two members, both with a target value of 0.2, and X_2 has two members with target values of 0.1 and 0.3, then the two members of X_2 are ranked as 1 and 4, and the two members of X_1 are both ranked as 2.5. Let $n_i = |X_i|$ be the size of each population and R_i the sum of the ranks of each member of either population ($i = 1,2$). Then the U-value is defined as

$$U = \min_{i=1,2} \left[n_1 n_2 + \frac{n_i(n_i+1)}{2} - R_i \right]. \tag{A.2}$$

This U-value follows the normal distribution. The z-score is calculated by normalising the U-value

$$z = \frac{U - \mu_U}{\sigma_U} \tag{A.3}$$

with

$$\mu_U = \frac{n_1 n_2 - 1}{2} \tag{A.4}$$

$$\sigma_U \approx \sqrt{\frac{n_1 n_2}{12} \left((n_1 + n_2 + 1) - \frac{\sum_{k=1}^{K} (t_k^3 - t_k)}{(n_1 + n_2)(n_1 + n_2 - 1)} \right)} \tag{A.5}$$

where K is the number of ranks with ties and t_k is the k-th rank with a tie [Ter19]. The critical z-score for a p-value of 0.05 is -1.96.

An explanation of how Figure 6.6(a) is created illustrates this process. Table A.3 shows all the p-values under the Mann-Whitney U test. The p-values are categorised as *significantly different* if $p < 0.05$. Pairs and groups of data that are not significantly different are grouped with a horizontal bar in the critical difference diagram.

Table A.3: Matrix of the p-values in a pairwise Mann–Whitney U test between the performance of the populations trained in the PointMaze environment.

	OL1	OSi	Sg3	Cus	LSA	20S	Ran
Occupancy **L1**	1.0	0.031	**0.85**	**0.162**	**0.473**	0.006	0.0
Occupancy **Si**nkhorn	0.031	1.0	0.045	0.045	0.031	**0.14**	**0.571**
Signature Depth **3**	**0.85**	0.045	1.0	**0.385**	**0.678**	**0.054**	0.002
Custom BD	**0.162**	0.045	**0.385**	1.0	**0.571**	**0.076**	0.001
Last **S**tate-**A**ction	**0.473**	0.031	**0.678**	**0.571**	1.0	0.031	0.0
20 Snapshots	0.006	**0.14**	**0.054**	**0.076**	0.031	1.0	0.006
Random BD	0.0	**0.571**	0.002	0.001	0.0	0.006	1.0

Random BD and *Occupancy Sinkhorn* form a group, as *Random BD* is significantly different from all other groups but *Occupancy Sinkhorn*. *Occupancy Sinkhorn*, however is not significantly different from *20 Snapshots* either, so the two form another group. *20 Snapshots* is further not significantly different from *Signature Depth 3* and *Custom BD*, which are also not significantly different from each other, which makes them a group of three. Finally, *Occupancy L1*, *Signature Depth 3*, *Custom BD*, and *Last State-Action* are all mutually not significantly different, so they also constitute a group.

To illustrate the calculation of the p-values, consider as an example the p-value of the comparison of *Custom BD* and *Occupancy Sinkhorn* in Table A.3.

Table A.4: The observed values of the best performance in a population of agents in the PointMaze environment and the respective ranks. Refers to Table A.3 and Figure 6.6(a).

Random BD	Rank	Occupancy Sinkhorn	Rank
-125.6	15	-59.67	9
-23.59	3	-22.66	2
-29.2	7	-25.85	6
-125.61	16	-125.790	20
-125.59	14	-58.56	8
-23.82	5	-22.59	1
-125.58	13	-125.68	18
-125.65	17	-65.98	10
-125.51	12	-23.77	4
-96.39	11	-125.789	19

The sum of ranks of the first group is $R_1 = 113$, the sum of ranks of the second group is $R_2 = 97$. That means, $U = 42$, $\mu = 49.5$ and $\sigma \approx 13.23$. The z-value is $z = \frac{U-\mu}{\sigma} \approx -0.57$. Consulting a z-table, this translates to a p-value of ≈ 0.571.

Danksagung

Danke an Prof. Dr. Jochen Garcke, meinen Doktorvater, für sein Vertrauen, für seine konstante Unterstützung, für die wissenschaftliche Zusammenarbeit und dafür, dass er sich immer und mit ganzer Kraft für das Wachstum und Wohlergehen der Menschen in seinem Umfeld engagiert.

Danke an Prof. Dr. Ira Neitzel für die Begutachtung dieser Arbeit und an die Mitglieder der Promotionskommission.

Danke an meine aktuellen und ehemaligen Kolleginnen und Kollegen am Fraunhofer SCAI, besonders an Daniela Steffes-lai und Daniel Oeltz für ihr offenes Ohr und für hilfreiche Impulse und an Sara Hahner und Christian Gscheidle, meine Kommilitonen, für ihre freundschaftliche Begleitung und kollegiale Unterstützung.

Danke an Biagio Paparella für die inspirierende Zusammenarbeit und an Paolo Climaco, Alexander Hagg und Erik Goodman für wertvolle fachliche Gespräche.

Danke an Kathrin Viertel, Kathrin Wolff und die SCAI IT für die unersätzliche Unterstützung bei allen Problemen, die neben der fachlichen Arbeit anfallen.

Ein besonderer Dank für das Korrekturlesen von Teilen dieser Arbeit gilt Bastian Bohn, Ivan Lecei, Jannis Dörhöfer, Ludovico Scarton und Tarek Iraki.

Danke an meine lieben Freundinnen und Freunde, Alexandra, Antonia, Danai, Felix, Franziska, Harriet, Jamie, Johannes, Jonathan, Johnny, Julia, Laura, Lisa, Lisa, Nathalie, Nicole, Philip, Raoul, Roberto, Tabea, Valentin und Vanessa, die mit Geduld und Verständnis bei mir waren.

Ich danke meinen Eltern, Irene und Bernd, für ihr bedingungsloses Vertrauen, für die Opfer, die sie gebracht haben, und für die Neugierde, die sie in mir gefördert haben.

Ich danke von ganzem Herzen Hannah, meiner Lebenspartnerin, für ihren Humor und Mut und für ihre Liebe und Unterstützung. Was ich auch tue, wo ich auch bin, mit Dir ist es immer besser.